Docket No. 193857US2/tsh

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

IN RE APPLICATION OF: Kazushi HONDA GAU:

SERIAL NO: 09/605,884 EXAMINER:

FILED: June 29, 2000

FOR: SYSTEM FOR OPTIMIZING DATA TYPE DEFINITION IN PROGRAM LANGUAGE PROCESSING, METHOD AND COMPUTER READABLE RECORDING MEDIUM THEREFOR

REQUEST FOR PRIORITY

ASSISTANT COMMISSIONER FOR PATENTS
WASHINGTON, D.C. 20231

SIR:

☐ Full benefit of the filing date of U.S. Application Serial Number [US App No], filed [US App Dt], is claimed pursuant to the provisions of **35 U.S.C. §120.**

☐ Full benefit of the filing date of U.S. Provisional Application Serial Number , filed , is claimed pursuant to the provisions of **35 U.S.C. §119(e).**

■ Applicants claim any right to priority from any earlier filed applications to which they may be entitled pursuant to the provisions of **35 U.S.C. §119**, as noted below.

In the matter of the above-identified application for patent, notice is hereby given that the applicants claim as priority:

COUNTRY	APPLICATION NUMBER	MONTH/DAY/YEAR
JAPAN	11-184252	June 29, 1999

Certified copies of the corresponding Convention Application(s)

■ are submitted herewith

☐ will be submitted prior to payment of the Final Fee

☐ were filed in prior application Serial No. filed

☐ were submitted to the International Bureau in PCT Application Number .
Receipt of the certified copies by the International Bureau in a timely manner under PCT Rule 17.1(a) has been acknowledged as evidenced by the attached PCT/IB/304.

☐ (A) Application Serial No.(s) were filed in prior application Serial No. filed ; and

(B) Application Serial No.(s)

☐ are submitted herewith

☐ will be submitted prior to payment of the Final Fee

Respectfully Submitted,

OBLON, SPIVAK, McCLELLAND,
MAIER & NEUSTADT, P.C.

Marvin J. Spivak
Registration No. 24,913

Surinder Sachar
Registration No. 34,423

Fourth Floor
1755 Jefferson Davis Highway
Arlington, Virginia 22202
Tel. (703) 413-3000
Fax. (703) 413-2220
(OSMMN 11/98)

09/605,884

日 本 国 特 許 庁
PATENT OFFICE
JAPANESE GOVERNMENT

別紙添付の書類に記載されている事項は下記の出願書類に記載されてる事項と同一であることを証明する。
This is to certify that the annexed is a true copy of the following application as filed in this Office.

出 願 年 月 日
Date of Application: 1999年 6月29日

出 願 番 号
Application Number: 平成11年特許願第184252号

出 願 人
Applicant (s): 株式会社東芝

CERTIFIED COPY OF
PRIORITY DOCUMENT

2000年 6月29日

特 許 庁 長 官
Commissioner,
Patent Office 近 藤 隆

出証番号 出証特2000-3053750

【書類名】　特許願

【整理番号】　46A98Y069

【提出日】　平成11年 6月29日

【あて先】　特許庁長官殿

【国際特許分類】　G06F 9/00

【発明の名称】　プログラム言語処理システム、そのコード最適化方法、及び機械読み出し可能な記憶媒体

【請求項の数】　10

【発明者】

【住所又は居所】　神奈川県川崎市幸区小向東芝町１番地　株式会社東芝マイクロエレクトロニクスセンター内

【氏名】　本多　和史

【特許出願人】

【識別番号】　000003078

【氏名又は名称】　株式会社　東芝

【代理人】

【識別番号】　100083806

【弁理士】

【氏名又は名称】　三好　秀和

【電話番号】　03-3504-3075

【選任した代理人】

【識別番号】　100068342

【弁理士】

【氏名又は名称】　三好　保男

【選任した代理人】

【識別番号】　100100712

【弁理士】

【氏名又は名称】　岩▲崎▼　幸邦

【選任した代理人】

【識別番号】　100100929

【弁理士】

【氏名又は名称】　川又　澄雄

【選任した代理人】

【識別番号】　100108707

【弁理士】

【氏名又は名称】　中村　友之

【選任した代理人】

【識別番号】　100095500

【弁理士】

【氏名又は名称】　伊藤　正和

【選任した代理人】

【識別番号】　100101247

【弁理士】

【氏名又は名称】　高橋　俊一

【選任した代理人】

【識別番号】　100098327

【弁理士】

【氏名又は名称】　高松　俊雄

【手数料の表示】

【予納台帳番号】　001982

【納付金額】　21,000円

【提出物件の目録】

【物件名】　明細書　1

【物件名】　図面　1

【物件名】　要約書　1

【プルーフの要否】　要

【書類名】　明細書

【発明の名称】　プログラム言語処理システム、そのコード最適化方法、及び機械読み出し可能な記憶媒体

【特許請求の範囲】

【請求項１】　原始プログラムの前処理を行う前処理装置と、前処理後の原始プログラムをコンパイルする言語処理装置と、前記原始プログラムを前記前処理装置に翻訳単位で渡し、少なくとも前記前処理装置と前記言語処理装置の間の処理結果の受け渡しを制御するソフトウェア駆動装置とを有するプログラム言語処理システムにおいて、

前記前処理後の原始プログラム全てを走査して、不要なコードを削除し必要な定義のみ実体化するコード最適化処理を行う最適化処理装置を設け、

前記ソフトウェア駆動装置は、

前記前処理後の原始プログラム全てを走査するために、前処理後の原始プログラム全てを前記最適化処理装置に渡す手段と、

前記最適化処理装置による前記コード最適化処理後の各翻訳単位を前記言語処理装置に渡す手段とを有することを特徴とするプログラム言語処理システム。

【請求項２】　前記最適化処理装置は、

前記前処理後の原始プログラムの中から多相型の定義を検知する多相型定義検知手段と、

前記多相型定義検知手段によって検知された多相型定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第１の判定手段と、

前記多相型定義の定義情報が未だ登録されていないと前記第１の判定手段によって判定されたときには、前記多相型定義の定義情報を前記多相型定義情報テーブルに登録する第１の登録手段と、

前記多相型定義の定義情報が既に登録されていると前記第１の判定手段によって判定されたときには、前記多相型定義検知手段によって検知された多相型定義を前記前処理後の原始プログラムの中から削除する第１の削除手段と、

前記前処理後の原始プログラムの中から多相型定義の実体生成要求を検知する第１の要求検知手段と、

前記第１の要求検知手段によって検知された多相型定義の実体生成要求に対応した多相型定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第２の判定手段と、

前記多相型定義の実体が未だ生成されていないと前記第２の判定手段によって判定されたときには、前記多相型定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第１の実体生成手段と、

前記多相型定義の実体が既に生成されていると前記第２の判定手段によって判定されたときには、該多相型定義の実体の生成を禁止する第１の実体生成禁止手段とを

有することを特徴とする請求項１記載のプログラム言語処理システム。

【請求項３】　前記最適化処理装置は、

前記前処理後の原始プログラムの中から多相型の関数定義を検知する関数定義検知手段と、

前記関数定義検知手段によって検知された多相型関数定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第３の判定手段と、

前記多相型関数定義の定義情報が未だ登録されていないと前記第３の判定手段によって判定されたときは、前記多相型関数定義の定義情報を前記多相型定義情報テーブルに登録する第２の登録手段と、

前記多相型関数定義の定義情報が既に登録されていると前記第３の判定手段によって判定されたときには、前記関数定義検知手段によって検知された多相型関数定義を前記前処理後の原始プログラムの中から削除する第２の削除手段と、

前記前処理後の原始プログラムの中から多相型関数定義の実体生成要求を検知する第２の要求検知手段と、

前記第２の要求検知手段によって検知された多相型実体生成要求に対応した多相型関数定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第４の判定手段と、

前記多相型関数定義の実体が未だ生成されていないと前記第４の判定手段によって判定されたときには、前記多相型関数定義の実体を生成すると共に、この実

体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第２の実体生成手段と、

前記多相型の関数定義の実体は既に生成されていると前記第４の判定手段によって判定されたときには、前記多相型の関数定義の実体生成を禁止する第２の実体生成禁止手段とを

有することを特徴とする請求項１または請求項２記載のプログラム言語処理システム。

【請求項４】　前記最適化処理装置は、

前記前処理後の原始プログラムの中から、メンバ関数を保持している多相型を検知するメンバ関数検知手段と、

前記メンバ関数検知手段によって多相型のメンバ関数が検知された場合に、そのメンバ関数が使用されているかどうかを表す使用情報を検知する使用情報手段と、

前記使用情報を多相型定義情報テーブルに登録する第３の登録手段と、

前記多相型定義情報テーブルの情報を基に、実際に生成する多相型のメンバ関数定義の実体を決定する決定手段と、

前記決定手段の決定に基づいて、多相型のメンバ関数定義の実体を生成する第３の実体生成手段とを

有することを特徴とする請求項１乃至請求項３記載の言語処理システム。

【請求項５】　前記最適化処理装置は、

前記前処理後の原始プログラムの中から、データまたは関数の定義を検知する定義検知手段と、

前記定義検知手段で検知されたデータまたは関数の定義に関する定義情報を定義情報テーブルに登録する第３の登録手段と、

前記定義情報を基に重複したデータまたは関数の定義を削除する第３の削除手段とを

有することを特徴とする請求項１乃至請求項４記載の言語処理システム。

【請求項６】　前処理後の原始プログラム全てを走査してコード最適化処理を行う最適化処理装置と、前記最適化処理装置によるコード最適化処理後の各翻

訳単位をコンパイルする言語処理装置と、前記最適化処理装置に対して前処理後の原始プログラム全てを渡すと共に、前記言語処理装置に対して前記コード最適化処理後の各翻訳単位を渡すソフトウェア駆動装置とを有するプログラム言語処理システムのコード最適化方法であって、

前記コード最適化処理は、

前記前処理後の原始プログラムの中から多相型の定義を検知する多相型定義検知処理と、

前記多相型定義検知処理によって検知された多相型定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第１の判定処理と、

前記多相型定義の定義情報が未だ登録されていないと前記第１の判定処理によって判定されたときには、前記多相型定義の定義情報を前記多相型定義情報テーブルに登録する第１の登録処理と、

前記多相型定義の定義情報が既に登録されていると前記第１の判定処理によって判定されたときには、前記多相型定義検知処理によって検知された多相型定義を前記前処理後の原始プログラムの中から削除する第１の削除処理と、

前記前処理後の原始プログラムの中から多相型定義の実体生成要求を検知する第１の要求検知処理と、

前記第１の要求検知処理によって検知された多相型定義の実体生成要求に対応した多相型定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第２の判定処理と、

前記多相型定義の実体が未だ生成されていないと前記第２の判定処理によって判定されたときには、前記多相型定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第１の実体生成処理と、

前記多相型定義の実体が既に生成されていると前記第２の判定処理によって判定されたときには、前記多相型定義の実体生成を禁止する第１の実体生成禁止処理とを

順次実行することを特徴とするプログラム言語処理システムのコード最適化方法。

【請求項７】　前記コード最適化処理は、

前記前処理後の原始プログラムの中から多相型の関数定義を検知する関数定義検知処理と、

前記関数定義検知処理によって検知された多相型関数定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第３の判定処理と、

前記多相型関数定義の定義情報が未だ登録されていないと前記第３の判定処理によって判定されたときは、前記多相型関数定義の定義情報を前記多相型定義情報テーブルに登録する第２の登録処理と、

前記多相型関数定義の定義情報が既に登録されていると前記第３の判定処理によって判定されたときには、前記関数定義検知処理によって検知された多相型関数定義を前記前処理後の原始プログラムの中から削除する第２の削除処理と、

前記前処理後の原始プログラムの中から多相型関数定義の実体生成要求を検知する第２の要求検知処理と、

前記第２の要求検知処理によって検知された多相型実体生成要求に対応した多相型関数定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第４の判定処理と、

前記多相型関数定義の実体が未だ生成されていないと前記第４の判定処理によって判定されたときには、前記多相型関数定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第２の実体生成処理と、

前記多相型の関数定義の実体は既に生成されていると前記第４の判定処理によって判定されたときには、前記多相型の関数定義の実体生成を禁止する第２の実体生成禁止処理とを

順次実行することを特徴とする請求項６記載のプログラム言語処理システムのコード最適化方法。

【請求項８】　前記コード最適化処理は、

前記前処理後の原始プログラムの中から、メンバ関数を保持している多相型を検知するメンバ関数検知処理と、

前記メンバ関数検知処理によって多相型のメンバ関数が検知された場合に、そ

のメンバ関数が使用されているかどうかを表す使用情報を検知する使用情報処理と、

前記使用情報を多相型定義情報テーブルに登録する第３の登録処理と、

前記多相型定義情報テーブルの情報を基に、実際に生成する多相型のメンバ関数定義の実体を決定する決定処理と、

前記決定処理の決定に基づいて、多相型のメンバ関数定義の実体を生成する第３の実体生成処理とを

順次実行することを特徴とする請求項６または請求項７記載の言語処理システムのコード最適化方法。

【請求項９】　前記コード最適化処理は、

前記前処理後の原始プログラムの中から、データまたは関数の定義を検知する定義検知処理と、

前記定義検知処理で検知されたデータまたは関数の定義に関する定義情報を定義情報テーブルに登録する第３の登録処理と、

前記定義情報を基に重複したデータまたは関数の定義を削除する第３の削除処理とを

順次実行することを特徴とする請求項６乃至請求項８記載の言語処理システムのコード最適化方法。

【請求項１０】　請求項６乃至請求項９記載の言語処理システムのコード最適化方法における前記コード最適化処理を実行するプログラムを格納したことを特徴とする機械読み出し可能な記憶媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】

本発明は、原始プログラムから目的プログラムを生成する等の機能を有するプログラム言語処理システム、そのコード最適化方法、及びこのコード最適化方法を実現するための機械読み出し可能な記憶媒体に関する。

【０００２】

【従来の技術】

プログラミング言語で扱うデータは、その採りうる値や操作の適用性などにより様々な型に分類されている。例えば、整数型や実数型、単精度型、倍精度型などの基本データ型のほか、同一または異なる同一、または異なる型で構成される型で構成される構造体を単位とした型も知られている。

【０００３】

このように、プログラミング言語の多くは型の概念を導入しており、その中でも、型をパラメータ化することにより、一度表現したアルゴリズムを様々な型に適用できる多相型が注目されている。多相型は、型の族を示すもので、型をパラメータとする型変数を導入することにより、整数型や実数型などという個別の型に依存しない型を定義するものであり、効率的なプログラミングの実現を可能にするものである。

【０００４】

図１１は、多相型の定義の従来例とその利用方法を示す図であり、図１２は、実体化された多相型の定義の従来例を示す図である。

【０００５】

図１１中に例示する定義１０１が多相型の定義である。１０２で示される「Ｔ」が型をパラメータとする型変数の宣言であり、１０３に示すものが実際の型変数である。この多相型の定義１０１より型変数に適当な型を適用すると、個々の型に応じた実際の定義が生成される。

【０００６】

図１１に示す１０４と１０５が多相型ｖｅｃｔｏｒのそれぞれｉｎｔ型（整数型）、ｄｏｕｂｌｅ型（倍精度型）に応じた定義の生成を要求しており、図１２の１０６，１０７に示すような定義の実体が後述の言語処理システムにおいて生成される。

【０００７】

図１３は、多相型の関数定義の従来例を示す図である。

【０００８】

同図の１０８で示される定義は、多相型の関数を定義している。これは個別の型に依存しない関数を意味する。また、抽象データ型を多相型に適用した場合、

そのデータを扱う関数を特に多相型のメンバ関数と呼ぶ。図１３の１０９で示される定義は、多相型のメンバ関数を定義している。

【０００９】

図１４は、従来のプログラム言語処理システムの構成を示すブロック図であり、図１５は、その処理の流れを示す概念図である。

【００１０】

このプログラム言語処理システムは、原始プログラムから目的プログラムを生成すべく、図１４に示すように、ソフトウェア駆動装置２０１、前処理装置２０３、言語処理装置２０４、及びリンク処理装置２０５で構成されている。

【００１１】

ソフトウェア駆動装置２０１は、原始プログラム群２０２を取り込み、これを前処理装置２０３、言語処理装置２０４、及びリンク処理装置２０５に順番に渡すような制御を行い、出力ファイルとして目的プログラムを得ることができる。

即ち、図１５に示すように、各原始プログラム２０２が対応する前処理装置２０３にそれぞれ渡されると、前処理が実行されて後段の言語処理装置２０４に解釈できるような形式のプログラム２０７がそれぞれ得られる。言語処理装置２０４は、この各前処理後のプログラム２０７に対してそれぞれコンパイルを実行し、各リロケータブルオブジェクトファイル２０８をリンク処理装置２０５へ出力する。そして、リンク処理装置２０５が、入力された各リロケータブルオブジェクトファイルをリンクし、これによって目的プログラム２０６が得られるのである。

【００１２】

かかるプログラム言語処理システムにおいて多相型が使用されたときは、その多相型の定義の実体を生成し展開することにより多相型のルーチンを実現している。

【００１３】

例えば、図１６に示す多相型の実現例では、多相型のデータやインラインコードが各翻訳単位毎に必要であるため、多相型が使用された時点３１０で、機械的に多相型の定義の実体３１１を翻訳単位に挿入し、多相型名を、指定された型を

基に図中の３１２に示すような適当なルールで生成されたユニークな文字列に変換することで、図１６では多相型の定義からｉｎｔ型の定義を実現している。

【００１４】

【発明が解決しようとする課題】

しかしながら、上記従来のプログラム言語処理システムでは、多相型の関数定義の実体を生成する場合に主に以下の問題点があった。

【００１５】

（１）多相型の関数定義を検知する場合の問題点

多相型の関数の場合では、定義の実体生成が要求される際に、その定義を言語処理システムが検知することは非常に困難である。具体的に説明すると、図１７に示すような、多相型の宣言ファイル（ヘッダファイル）ｓｔａｃｋ．ｈ、多相型の定義ファイルｓｔａｃｋ．ｃｐｐ、及び多相型を使用するソースファイルｆｉｌｅ．ｃｐｐを有する標準的なファイル構成の場合においては、インクルード指令３１３，３１４により、ファイルｓｔａｃｋ．ｈにある多相型の定義の取り込みが行われるが、言語処理システムが多相型の関数定義を検知することができないため、ファイルｓｔａｃｋ．ｃｐｐに存在する多相型のメンバ関数定義は、ファイルｆｉｌｅ．ｃｐｐには取り込まれないことになる。

【００１６】

ここで、宣言３１５により、多相型定義の実体要求が発生する。しかし、ファイルｆｉｌｅ．ｃｐｐには翻訳単位中に多相型のメンバ関数定義は存在しないため、言語処理システムは多相型のメンバ関数定義の実体を生成しようと試みるが、定義が別ファイルｓｔａｃｋ．ｃｐｐに存在するため、その多相型のメンバ関数定義の実体を生成することができない状況に陥る。

【００１７】

したがって、従来では、多相型の関数定義を検索するために、多相型関数定義をヘッダファイルに置くことを要求したり（第１の対応策）、または多相型関数定義が置かれるファイル名に制限を設ける（第２の対応策）ことなどで対応している。

【００１８】

前記第１の対応策では、図１８に例示されるように、ヘッダファイルｓｔａｃｋ．ｈ内に、多相型の宣言だけではなく、３１６で示された全ての定義を記述されることが要求されている。したがって、３１８でソースファイルｆｉｌｅ．ｃｐｐにある多相型の実体を要求した際に、３１７で示されるようにそのヘッダファイルがインクルードされていれば、言語処理システムは多相型の関数定義をその翻訳単位で検知することが可能となる。しかしこの場合は、多相型の関数定義を変更する際に必ず再コンパイルを必要とするため、ファイル構成に柔軟さを欠いている。

【００１９】

前記第２の対応策では、言語処理システムが多相型の関数定義を検知するために、多相型の関数定義が置かれるファイル名を、言語処理システムが多相型の関数定義を検知するために、ある規則に従った名前（多くは多相型名と同一）にすることを要求する。

【００２０】

例えば図１９に示すように、ヘッダファイルｓｔａｃｋ．ｈに置かれた多相型Ｓｔａｃｋのメンバ関数の定義は、その多相型名Ｓｔａｃｋと同じソースファイル名Ｓｔａｃｋ．ｃｐｐに置くことを原始プログラム作成者に要求する。これによって、宣言３２０により、３１９で示される多相型の関数定義「Ｓｔａｃｋ：：Ｓｔａｃｋ」の実体生成が要求されたときに、決められた名前のファイルＳｔａｃｋ．ｃｐｐ中にその定義を検索し、多相型の関数定義の実体を生成することを可能としている。

【００２１】

しかし、この場合は複数の多相型定義をヘッダに記述することができず、多相型の数だけ制限された名前のファイルを作成する必要性が生じ、不必要な手間が原始プログラム作成者に要求される。

【００２２】

（２）不必要な多相型のメンバ関数定義の実体が生成される場合

多相型のメンバ関数の場合、利用されないメンバ関数の実体を生成することはコンパイル時間を増加させ、メモリ領域を余計に使用することになる。具体的に

説明すると、図２０で例示されるような多相型の定義の場合、３２１で示される多相型のメンバ関数の数が１００個で、実際に利用される多相型のメンバ関数の数が１つだけの場合には、残り９９個の多相型のメンバ関数定義の実体を生成することは明らか効率が悪いことが分かる。

【００２３】

したがって、従来では、効率を無視して全ての多相型のメンバ関数定義の実体を生成するか、利用されている多相型のメンバ関数が全て分かるまで、多相型のメンバ関数定義の実体生成を遅らせることで対応している。

【００２４】

（３）複数のリロケータブルオブジェクトファイルに多相型の関数定義の実体が生成される場合

多相型の関数定義は、多相型の関数が利用されるたびに実体生成の要求が起きる。したがって、生成要求毎に多相型の関数定義の実体を生成すると、複数の同じ定義が存在することになる。例えば図２１で示されるように、同一翻訳単位内３２２，３２３で多相型の関数定義の実体を生成する場合には、言語処理システムがその多相型の関数定義が実体化されたかどうかのフラグ変数を持つことで、重複した定義の生成を回避することが可能である。

【００２５】

しかし、別の翻訳単位で３２４に示されるような多相型の関数定義の実体生成要求がある場合には、従来の言語処理システムが他の翻訳単位で既にその多相型の関数定義が生成されているかどうかを判別することができないという問題がある。

【００２６】

したがって、従来では、原始プログラム作成者が特別な前処理指令を用いて重複した多相型の関数定義の実体生成を抑制する方法や、リンク処理時にリンクが重複した多相型の関数定義を削除する方法で対応しているが、前者は原始プログラム作成者に不要な制限を要求しており、後者はその判別処理に多大な時間を必要としている。

【００２７】

本発明は、上述の如き従来の問題点を解決するためになされたもので、その目的は、多相型の関数定義を容易に検知することができ、しかも重複した不要な多相型の定義を取り除くことができると共に、不要な多相型の実体化を回避することができるプログラム言語処理システム、そのコード最適化方法、及びこのコード最適化方法を実現するための機械読み出し可能な記憶媒体を提供することである。

【００２８】

【課題を解決するための手段】

上記目的を達成するために、請求項１記載の発明に係るプログラム言語処理システムでは、原始プログラムの前処理を行う前処理装置と、前処理後の原始プログラムをコンパイルする言語処理装置と、前記原始プログラムを前記前処理装置に翻訳単位で渡し、少なくとも前記前処理装置と前記言語処理装置の間の処理結果の受け渡しを制御するソフトウェア駆動装置とを有するプログラム言語処理システムにおいて、前記前処理後の原始プログラム全てを走査して、不要なコードを削除し必要な定義のみ実体化するコード最適化処理を行う最適化処理装置を設け、前記ソフトウェア駆動装置は、前記前処理後の原始プログラム全てを走査するために、前処理後の原始プログラム全てを前記最適化処理装置に渡す手段と、前記最適化処理装置による前記コード最適化処理後の各翻訳単位を前記言語処理装置に渡す手段とを有することを特徴とする。

【００２９】

請求項２記載の発明に係るプログラム言語処理システムでは、請求項１記載のプログラム言語処理システムにおいて、前記最適化処理装置は、前記前処理後の原始プログラムの中から多相型の定義を検知する多相型定義検知手段と、前記多相型定義検知手段によって検知された多相型定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第１の判定手段と、前記多相型定義の定義情報が未だ登録されていないと前記第１の判定手段によって判定されたときには、前記多相型定義の定義情報を前記多相型定義情報テーブルに登録する第１の登録手段と、前記多相型定義の定義情報が既に登録されていると前記第１の判定手段によって判定されたときには、前記多相型定義検知手段によって検知

された多相型定義を前記前処理後の原始プログラムの中から削除する第１の削除手段と、前記前処理後の原始プログラムの中から多相型定義の実体生成要求を検知する第１の要求検知手段と、前記第１の要求検知手段によって検知された多相型定義の実体生成要求に対応した多相型定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第２の判定手段と、前記多相型定義の実体が未だ生成されていないと前記第２の判定手段によって判定されたときには、前記多相型定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第１の実体生成手段と、前記多相型定義の実体が既に生成されていると前記第２の判定手段によって判定されたときには、該多相型定義の実体生成を禁止する第１の実体生成禁止手段とを有することを特徴とする。

【００３０】

請求項３記載の発明に係るプログラム言語処理システムでは、請求項１または請求項２記載のプログラム言語処理システムにおいて、前記最適化処理装置は、前記前処理後の原始プログラムの中から多相型の関数定義を検知する関数定義検知手段と、前記関数定義検知手段によって検知された多相型関数定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第３の判定手段と、前記多相型関数定義の定義情報が未だ登録されていないと前記第３の判定手段によって判定されたときは、前記多相型関数定義の定義情報を前記多相型定義情報テーブルに登録する第２の登録手段と、前記多相型関数定義の定義情報が既に登録されていると前記第３の判定手段によって判定されたときには、前記関数定義検知手段によって検知された多相型関数定義を前記前処理後の原始プログラムの中から削除する第２の削除手段と、前記前処理後の原始プログラムの中から多相型関数定義の実体生成要求を検知する第２の要求検知手段と、前記第２の要求検知手段によって検知された多相型実体生成要求に対応した多相型関数定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第４の判定手段と、前記多相型関数定義の実体が未だ生成されていないと前記第４の判定手段によって判定されたときには、前記多相型関数定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として

前記多相型定義情報テーブルに登録する第２の実体生成手段と、前記多相型の関数定義の実体は既に生成されていると前記第４の判定手段によって判定されたときには、前記多相型の関数定義の実体生成を禁止する第２の実体生成禁止手段とを有することを特徴とする。

【００３１】

請求項４記載の発明に係るプログラム言語処理システムでは、請求項１乃至請求項３記載の言語処理システムにおいて、前記最適化処理装置は、前記前処理後の原始プログラムの中から、メンバ関数を保持している多相型を検知するメンバ関数検知手段と、前記メンバ関数検知手段によって多相型のメンバ関数が検知された場合に、そのメンバ関数が使用されているかどうかを表す使用情報を検知する使用情報手段と、前記使用情報を多相型定義情報テーブルに登録する第３の登録手段と、前記多相型定義情報テーブルの情報を基に、実際に生成する多相型のメンバ関数定義の実体を決定する決定手段と、前記決定手段の決定に基づいて、多相型のメンバ関数定義の実体を生成する第３の実体生成手段とを有することを特徴とする。

【００３２】

請求項５記載の発明に係るプログラム言語処理システムでは、請求項１乃至請求項４記載の言語処理システムにおいて、前記最適化処理装置は、前記前処理後の原始プログラムの中から、データまたは関数の定義を検知する定義検知手段と、前記定義検知手段で検知されたデータまたは関数の定義に関する定義情報を定義情報テーブルに登録する第３の登録手段と、前記定義情報を基に重複したデータまたは関数の定義を削除する第３の削除手段とを有することを特徴とする。

【００３３】

請求項６記載の発明に係るプログラム言語処理システムのコード最適化方法では、前処理後の原始プログラム全てを走査してコード最適化処理を行う最適化処理装置と、前記最適化処理装置によるコード最適化処理後の各翻訳単位をコンパイルする言語処理装置と、前記最適化処理装置に対して前処理後の原始プログラム全てを渡すと共に、前記言語処理装置に対して前記コード最適化処理後の各翻訳単位を渡すソフトウェア駆動装置とを有するプログラム言語処理システムのコ

ード最適化方法であって、前記コード最適化処理は、前記前処理後の原始プログラムの中から多相型の定義を検知する多相型定義検知処理と、前記多相型定義検知処理によって検知された多相型定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第１の判定処理と、前記多相型定義の定義情報が未だ登録されていないと前記第１の判定処理によって判定されたときには、前記多相型定義の定義情報を前記多相型定義情報テーブルに登録する第１の登録処理と、前記多相型定義の定義情報が既に登録されていると前記第１の判定処理によって判定されたときには、前記多相型定義検知処理によって検知された多相型定義を前記前処理後の原始プログラムの中から削除する第１の削除処理と、前記前処理後の原始プログラムの中から多相型定義の実体生成要求を検知する第１の要求検知処理と、前記第１の要求検知処理によって検知された多相型定義の実体生成要求に対応した多相型定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第２の判定処理と、前記多相型定義の実体が未だ生成されていないと前記第２の判定処理によって判定されたときには、前記多相型定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第１の実体生成処理と、前記多相型定義の実体が既に生成されていると前記第２の判定処理によって判定されたときには、前記多相型定義の実体生成を禁止する第１の実体生成禁止処理とを順次実行することを特徴とする。

【００３４】

請求項７記載の発明に係るプログラム言語処理システムのコード化最適化方法では、請求項６記載のプログラム言語処理システムのコード化最適化方法において、前記コード最適化処理は、前記前処理後の原始プログラムの中から多相型の関数定義を検知する関数定義検知処理と、前記関数定義検知処理によって検知された多相型関数定義の定義情報が既に多相型定義情報テーブルに登録されているか否かを判定する第３の判定処理と、前記多相型関数定義の定義情報が未だ登録されていないと前記第３の判定処理によって判定されたときは、前記多相型関数定義の定義情報を前記多相型定義情報テーブルに登録する第２の登録処理と、前記多相型関数定義の定義情報が既に登録されていると前記第３の判定処理によっ

て判定されたときには、前記関数定義検知処理によって検知された多相型関数定義を前記前処理後の原始プログラムの中から削除する第２の削除処理と、前記前処理後の原始プログラムの中から多相型関数定義の実体生成要求を検知する第２の要求検知処理と、前記第２の要求検知処理によって検知された多相型実体生成要求に対応した多相型関数定義の実体が既に生成されているか否かを前記多相型定義情報テーブルの実体化情報を参照して判定する第４の判定処理と、前記多相型関数定義の実体が未だ生成されていないと前記第４の判定処理によって判定されたときには、前記多相型関数定義の実体を生成すると共に、この実体生成を表す情報を前記実体化情報として前記多相型定義情報テーブルに登録する第２の実体生成処理と、前記多相型の関数定義の実体は既に生成されていると前記第４の判定処理によって判定されたときには、前記多相型の関数定義の実体生成を禁止する第２の実体生成禁止処理とを順次実行することを特徴とする。

【００３５】

請求項８記載の発明に係るプログラム言語処理システムのコード化最適化方法では、請求項６または請求項７記載の言語処理システムのコード最適化方法において、前記コード最適化処理は、前記前処理後の原始プログラムの中から、メンバ関数を保持している多相型を検知するメンバ関数検知処理と、前記メンバ関数検知処理によって多相型のメンバ関数が検知された場合に、そのメンバ関数が使用されているかどうかを表す使用情報を検知する使用情報処理と、前記使用情報を多相型定義情報テーブルに登録する第３の登録処理と、前記多相型定義情報テーブルの情報を基に、実際に生成する多相型のメンバ関数定義の実体を決定する決定処理と、前記決定処理の決定に基づいて、多相型のメンバ関数定義の実体を生成する第３の実体生成処理とを順次実行することを特徴とする。

【００３６】

請求項９記載の発明に係るプログラム言語処理システムのコード化最適化方法では、請求項６乃至請求項８記載の言語処理システムのコード最適化方法において、前記コード最適化処理は、前記前処理後の原始プログラムの中から、データまたは関数の定義を検知する定義検知処理と、前記定義検知処理で検知されたデータまたは関数の定義に関する定義情報を定義情報テーブルに登録する第３の登

録処理と、前記定義情報を基に重複したデータまたは関数の定義を削除する第３の削除処理とを順次実行することを特徴とする。

【００３７】

請求項１０記載の発明に係る機械読み出し可能な記憶媒体では、請求項６乃至請求項９記載の言語処理システムのコード最適化方法における前記コード最適化処理を実行するプログラムを格納したことを特徴とする。

【００３８】

【発明の実施の形態】

以下、本発明の実施の形態を図面に基づいて説明する。

【００３９】

［第１実施形態］

図１は、本発明の第１実施形態に係るプログラム言語処理システムの基本構成を示すブロック図である。

【００４０】

このプログラム言語処理システムは、原始プログラム群２から目的プログラム７を生成すべく、ソフトウェア駆動装置１、前処理装置３、最適化処理装置４、言語処理装置５、及びリンク処理装置６で構成されている。そのうち、前処理装置３、言語処理装置５及びリンク処理装置６は、図１４に示した従来システムにおける前処理装置２０３、言語処理装置２０４及びリンク処理装置２０５と同一の機能を有するものである。従って、本実施形態のプログラム言語処理システムの構成では、図１４の従来構成の言語処理システムにおいて、新たに最適化処理装置４を設け、さらにこれに対応して、ソフトウェア駆動装置２０１の機能を拡張したソフトウェア駆動装置１を設けた点が特徴となっている。

【００４１】

即ち、本実施形態のプログラム言語処理システムは、例えばコンパイラなど、ＣＰＵのオブジェクトコードを生成するシステムであり、多相型を扱うことができ、前処理が終了した全ての入力プログラムを最適化処理装置４に渡すことができるように制御可能なソフトウェア駆動装置１を有している。最適化処理装置４は、全ての入力プログラムに対し多相型定義情報テーブル４ａを作成し、そのテ

ーブル４ａに格納された情報により不要なコードを削除することで、コードの最適化を実施する。

【００４２】

以下、本実施形態の特徴部分について詳細に説明する。

【００４３】

図２は、本実施形態に係るプログラム言語処理システムの処理の流れを示す概念図である。

【００４４】

ソフトウェア駆動装置１は、取り込んだ原始プログラム群２を、前処理装置３、最適化処理装置４、言語処理装置５、及びリンク処理装置６に順番に渡すような制御を行う。

【００４５】

原始プログラム群２がソフトウェア駆動装置１により各ファイル毎に前処理装置３に渡されると、前処理装置３による前処理指令の実行により従来の言語処理システムと同様の前処理が行われる。その結果、前処理装置３からは、前処理後のファイル８が各翻訳単位として出力される。ここで、全ての前処理後のファイル８が最適化処理装置４に入力され、後述するコード最適化処理が全ての入力ファイルに対して完了した後に、各々の翻訳単位がパラレルに言語処理装置５に振り分けられる。ここで、最適化処理装置４への前処理後の全ファイルの読み込み制御と、言語処理装置５への最適化処理後のファイルの振り分け制御は、ソフトウェア駆動装置１によって行われ、このソフトウェア駆動装置１の機能が従来システムのソフトウェア駆動装置２０１に対して拡張された機能となっている。

【００４６】

以降の処理は、従来の言語処理システムと同様に各言語処理装置５が生成したリロケータブルオブジェクト９をリンク処理装置６でリンクして、目的プログラム７が得られる。

【００４７】

次に、本実施形態のコード最適化処理を図３、図４及び図５を参照して説明する。なお、図３は、本実施形態に係るコード最適化処理を示すフローチャートで

ある。また、図４は、本実施形態に係る、多相型定義情報テーブル４ａへの多相型定義の登録例を示す図であり、図５は、本実施形態に係る多相型定義の実体生成例を示す図である。

【００４８】

図３において、まず、前述したように、ソフトウェア駆動装置１が原始プログラム２を前処理装置３に渡し（ステップＳ１１）、前処理装置３が前処理指令を実行して前処理が行われると（ステップＳ１２）、ソフトウェア駆動装置１は、前処理後のプログラムｆ０を最適化処理装置４に渡す（ステップＳ１３）。

【００４９】

最適化処理装置４は、コードの最適化を実行するために、今回入力された前処理後のプログラムｆ０を走査し、当該入力プログラムｆ０中の多相型の定義を検索する（ステップＳ１４）。このとき、多相型定義が存在しないときは、後述するステップＳ１８の多相型実体要求の検索処理に移行する。

【００５０】

多相型の定義が存在すれば、その定義が既にあるかどうかを多相型定義情報テーブル４ａの定義情報を基に調べ（ステップＳ１５）、未だ定義されていない多相型定義であれば、多相型定義情報テーブル４ａにその定義情報を登録する（ステップＳ１６）。

【００５１】

この多相型定義情報テーブル４ａへの登録処理では、検知した多相型定義の型名「Ｔｒｅｅ」３８を定義情報として、多相型定義情報テーブル４ａに登録する（図４の４０参照）。そして、このとき、多相型「Ｔｒｅｅ」の定義が今回のプログラムｆ０上で使用されていれば（図４中の３９参照）、多相型定義情報テーブル４ａ上の多相型定義情報の使用フラグを使用状態に設定する（図４の４１参照）。

【００５２】

一方、既に定義されている多相型の定義であれば、その定義は重複していることになるため、多相型定義情報テーブル４ａにはその定義情報を登録せず、重複した当該多相型の定義を入力プログラムｆ０中から削除する（ステップＳ１７）

。即ち、既に、型名「Ｔｒｅｅ」が多相型定義情報テーブル４ａに登録されていれば、今回検知された多相型定義３７は重複した定義とみなし、今回読み込まれた入力プログラムｆ０上から削除するのである。

【００５３】

また、図３で示した本実施形態のコード最適化処理の実行時に、多相型定義情報テーブル４ａに登録された型名で、その型名が全プログラム中で使用されていないと判明したときは（前記使用フラグが未使用状態）、最適化処理装置４は、当該多相型定義を不要とみなし、入力プログラム上から該多相型定義を削除する。　かようにして多相型定義の走査が完了した後、最適化処理装置４は、多相型の使用による多相型定義の実体化要求を検索する（ステップＳ１８）。このとき、多相型定義の実体化要求が存在すれば、その実体化された多相型定義の型が既に存在するかどうかを、多相型定義情報テーブル４ａを基に調べ（ステップＳ１９）、未だ実体化されていない多相型定義であれば、多相型の実体（定義）を生成する（ステップＳ２０）。一方、既に実体化された多相型定義であれば、その実体化された定義は重複することになるため、多相型の実体は生成しない（ステップＳ２１）。

【００５４】

この多相型実体生成処理について図５を用いて具体的に説明する。最適化処理装置４が、例えばｉｎｔ型を使用する多相型実体生成要求４５を検知すると、その多相型定義の実体が既に生成されているかどうかを多相型定義情報テーブル４ａに問い合わせる。このとき、未だ実体が生成されていなければ、４３で示すｉｎｔ型の多相型定義の実体を生成し、多相型定義情報テーブル４ａには既に実体を生成した旨の情報が登録される。この際、多相型の名前は、プログラム中でユニークな名前に変換され、ｉｎｔ型を使用する多相型実体生成要求４５などの全ての型名も、４８で示すようにユニークな名前に変換される。

【００５５】

同様に、ｄｏｕｂｌｅ型を使用する多相型実体生成要求４６が検知されると、４４で示すｄｏｕｂｌｅ型の多相型定義の実体生成、多相型定義情報テーブル４ａへの情報登録、及び４９，５０で示すようなユニークな名前への変換が行われ

る。ここで、４７で示すような、ｄｏｕｂｌｅ型を使用する多相型定義の実体生成要求を検知した場合は、多相型定義情報テーブル４ａには「既にｄｏｕｂｌｅ型の多相型定義の実体は生成済み」との情報があるため、この情報に基づき４４で示すようなｄｏｕｂｌｅ型を使用する多相型定義の実体は生成しない。

【００５６】

また、多相型定義の実体を生成する際に、もしｉｎｔ型とｄｏｕｂｌｅ型が同じ型のアーキテクチャであるような場合には、多相型定義の実体は同じものとみなされ、多相型定義の実体は１つしか生成されない。以上の結果、図５に示すような最適化後の原始プログラムｆ０’が得られる。

【００５７】

これ以降は、上記の処理を全ての原始プログラムに対して繰り返し（ステップＳ２２）、処理が終了した各プログラムを、ソフトウェア駆動装置１により言語処理装置５に渡すことになる（ステップＳ２３）。

【００５８】

このように本実施形態では、最適化処理装置４が、上記コード最適化処理を全ての原始プログラムに対して実行するので、全ての原始プログラム内の多相型の定義情報と多相型の実体化要求情報とを収集することができる。これにより、重複した不要な多相型定義を取り除くことができると共に、不要な多相型の実体生成を回避することができるため、オブジェクトサイズを小さくし、実行速度を向上させることが可能になる。また、コンパイル以前に最適化を完了するため、多相型にかかるコンパイル・リンク時間の高速化が図られる。

【００５９】

［第２実施形態］

第２実施形態では、図１、図２、及び図３で説明した上記第１実施形態の構成において、多相型の関数定義に関して、特に多相型定義情報テーブルへの登録、及び実体生成について説明するものである。

【００６０】

図６は、本実施形態に係る多相型関数定義の多相型定義情報テーブル４ａへの登録例を説明するための図である。

【００６１】

最適化処理装置４は、前処理後の原始プログラムｆ１とｆ２を読み込み（図３のステップＳ１３）、多相型の関数定義５１を検知すると（図３のステップＳ１４）、その関数定義５１が既にあるかどうかを多相型定義情報テーブル４ａの定義情報を基に調べる（図３のステップＳ１５）。

【００６２】

未だ定義されていない多相型の関数定義であれば、検知した多相型の型名「ｍａｘ」５２を多相型定義情報テーブル４ａに登録する。もしこのとき、既に型名「ｍａｘ」５２が多相型定義情報テーブル４ａに登録されていれば、この多相型の関数定義５１は重複した定義とみなし、前処理後の原始プログラムｆ１上から削除する（図３のステップＳ１７）。

【００６３】

さらに、図６の５３，５４，５５のように多相型５１が使用されていれば、それぞれ多相型定義の実体要求が生じた型毎に、５６，５７で示される多相型定義情報テーブル４ａ上の多相型定義情報の使用フラグを使用状態に設定する。

【００６４】

また、多相型定義情報テーブル４ａに登録された多相型の関数名で、その関数名が全プログラム中で使用されていなければ、最適化処理装置４はその多相型の定義を不要とみなし、前処理後の原始プログラム上から定義を削除する。

【００６５】

図７は、本実施形態に係る多相型関数定義の実体生成例を示す図である。

【００６６】

最適化処理装置４は、図６で説明したように多相型の関数定義５１を読み込み、多相型定義情報テーブル４ａに登録する。その後、ｉｎｔ型を使用する多相型の関数定義の実体生成要求５３を検知すると（図３のステップＳ１８）、多相型定義情報テーブル４ａにその多相型の関数定義の実体が既に生成されているかどうか問い合わせる（図３のステップＳ１９）。

【００６７】

このとき未だ実体が生成されていなければ、６２で示すｉｎｔ型を使用する多

相型の関数定義の実体を生成し（図３のステップＳ２０）、多相型定義情報テーブル４ａには既に実体を生成した旨の情報が登録される。この際、多相型関数の名前はプログラム中でユニークな名前に変換され、その関数を使用する６４，６６などに示す全ての関数名がそのユニークな名前に変換される。

【００６８】

同様に、ｄｏｕｂｌｅ型を使用する多相型の関数定義の実体生成要求５４が検知されると、６３で示すｄｏｕｂｌｅ型を使用する多相型の関数定義の実体を生成し（図３のステップＳ２０に相当）、多相型定義情報テーブル４ａへの実体生成済みとの情報登録と、６５で示すようなユニークな名前への変換とが行われる。

【００６９】

ここで、６１で示すｉｎｔ型を使用する多相型の関数定義の実体生成要求を検知した場合、多相型定義情報テーブル４ａには、ｉｎｔ型を使用する多相型の関数定義の実体は既に生成済みとの情報があるため、６２で示すようなｉｎｔ型を使用する多相型の関数定義の実体を生成することは行わない（図３のステップＳ２１）。

【００７０】

また、多相型関数の定義の実体を生成する際に、もしｉｎｔ型とｄｏｕｂｌｅ型が同じ型のアーキテクチャであるような場合には、多相型関数の定義の実体は同じものとみなされ、多相型関数の定義の実体は１つしか生成されない。

【００７１】

以上の結果、図７に示すような最適化後の原始プログラムｆ１’，ｆ２’が得られる。

【００７２】

このように、本実施形態においても、多相型の関数定義において、上記第１実施形態と同等の効果を得ることができる。

【００７３】

［第３実施形態］

第３実施形態では、図１、図２、及び図３で説明した上記第１実施形態の構成

において、多相型のメンバ関数定義に関して、特に多相型定義情報テーブルへの登録、及び実体生成について説明するものである。

【００７４】

図８は、本実施形態に係る多相型メンバ関数定義の多相型定義情報テーブル４ａへの登録例を示す図である。また、図９は、本実施形態に係る多相型メンバ関数の最適化処理を示すフローチャートであり、この処理は、図３におけるステップＳ２０における多相型の実体（定義）生成処理時に実施される。

【００７５】

最適化処理装置４は、図８に示された前処理後の原始プログラムｆ３を読み込み（図３のステップＳ１３）、多相型の定義６６を検知する（図３のステップＳ１４）。このとき、多相型がメンバ関数を持つならば、最適化処理装置４は、６７に示す多相型のメンバ関数が６８，６９のように使用されているかの情報を同時に検知し（図３のステップＳ１５）、多相型のメンバ関数使用情報を７０，７１のように多相型定義情報テーブル４ａに登録し（図３のステップＳ１６）、７２のように使用フラグを使用状態に設定する。

【００７６】

その後、最適化処理装置４が、多相型の実体生成要求７６を検知すると（図３のステップＳ１８）、多相型定義情報テーブル４ａに問い合わせ、未だ当該多相型定義の実体が生成されていなければ（図３のステップＳ１９）、この多相型定義の実体（定義）生成を行う（ステップＳ２０）。

【００７７】

この多相型の実体（定義）の生成時に、最適化処理装置４は、図９のフローチャートに示す処理を実施する。図９において、まず全ての多相型のメンバ関数で必要となるメンバ変数定義の実体化を行う（ステップＳ３１）。

【００７８】

次いで、多相型がメンバ関数を持つならば、そのメンバ関数が使用されているかどうかを多相型定義情報テーブル４ａに問い合わせ（ステップＳ３２，ステップＳ３３）、使用されている場合はその多相型メンバ関数定義の実体を生成する（ステップＳ３４）。また使用されていない場合は、その多相型メンバ関数の定

義の実体は生成しない。

【００７９】

この処理を全ての多相型のメンバ関数について調べる（ステップＳ３５）。これにより、本実施形態では、使用されていない多相型のメンバ関数定義の実体を生成するのを未然に回避することができる。

【００８０】

図１０は、本実施形態に係る多相型のメンバ関数定義の実体生成例を示す図である。

【００８１】

最適化処理装置４が、例えばｉｎｔ型の多相型実体生成要求７６を検知すると、多相型定義情報テーブル４ａに問い合わせ、未だ当該多相型定義の実体が生成されていなければ、多相型のメンバ関数の使用情報を基に、実際に生成する多相型のメンバ関数を決定する。

【００８２】

図１０に示した例では、ｉｎｔ型の実体で使用される多相型のメンバ関数定義７４は２つであるため、実際にｉｎｔ型の多相型定義７８で実体が生成されるメンバ関数は７９に示すようになり、他の使用されていない多相型のメンバ関数定義の実体は生成されない。

【００８３】

同様にｄｏｕｂｌｅ型の多相型の実体生成要求７７が検知されると、未だその多相型定義の実体が生成されていなければ、多相型定義情報テーブル４ａより多相型のメンバ関数の使用情報を基に実際に生成する多相型のメンバ関数７５が１つだけであると決定する。この決定に従い、実際にｄｏｕｂｌｅ型の多相型の定義８０で実体が生成されるメンバ関数は８１で示す１つだけとなる。

【００８４】

以上の結果、図１０に示すような最適化後の原始プログラムｆ３’が得られる。

【００８５】

このように、本実施形態においても、多相型のメンバ関数定義において、上記

第１実施形態と同等の効果を得ることができる。

【００８６】

なお、上記第１、第２、及び第３実施形態では、多相型で説明をしたが、図１及び図２に示した構成を利用することにより、多相型に限らず、原始プログラム群中における重複したデータ、重複した関数を削除することができる。

【００８７】

この場合も、多相型同様に最適化処理装置４が前処理後の原始プログラムを読み込み、各データ、関数の定義を定義情報テーブル４ａに登録する。このとき、定義情報テーブル４ａの情報を基に既に定義されているデータや関数を最適化処理装置４が検知した場合、このデータや関数の定義は重複した定義とみなし、前処理後のプログラムから削除する。また、定義情報テーブル４ａに登録されたデータや関数の定義で、そのデータや関数が全プログラム中で使用されていなければ、最適化処理装置４はその定義を不要とみなし、前処理後の原始プログラム上から定義を削除する。

【００８８】

また、上記第１、第２、及び第３実施形態の各機能を実現するためのプログラムをＣＤ－ＲＯＭなどの記憶媒体に格納することで、本発明を実現することも可能である。

【００８９】

【発明の効果】

以上詳細に説明したように、請求項１記載の発明によれば、不要なコードを削除し必要な定義のみ実体化することができるので、コードサイズを縮小することができ、実行速度が向上する。さらに、コンパイル以前にコード最適化処理が完了するため、コンパイルやリンク時間の高速化が実現する。

【００９０】

請求項２記載の発明によれば、請求項１の発明において、重複した不要な多相型の定義を取り除くことができると共に、不要な多相型の実体化を回避することができ、多相型に関して上記請求項１の発明と同等の効果を奏する。

【００９１】

請求項３記載の発明によれば、請求項１または請求項２の発明において、多相型の関数定義を容易に検知することができ、しかも重複した不要な多相型関数の定義を取り除くことができると共に、不要な多相型関数の実体化を回避することができ、多相型に関して上記請求項１の発明と同等の効果を奏する。

【００９２】

請求項４記載の発明によれば、請求項１乃至請求項３の発明において、重複した不要な多相型のメンバ関数定義を取り除くことができると共に、不要な多相型メンバ関数の実体化を回避することができ、多相型に関して上記請求項１の発明と同等の効果を奏する。

【００９３】

請求項５記載の発明によれば、請求項１乃至請求項４の発明において、データまたは関数の定義の重複を防ぐことができ、全てのデータ型に関して上記請求項１の発明と同等の効果を奏する。

【００９４】

請求項６記載の発明によれば、重複した不要な多相型の定義を取り除くことができると共に、不要な多相型の実体化を回避することができ、多相型に関して上記請求項１の発明と同等の効果を奏する。

【００９５】

請求項７記載の発明によれば、請求項６の発明において、多相型の関数定義を容易に検知することができ、しかも重複した不要な多相型関数の定義を取り除くことができると共に、不要な多相型関数の実体化を回避することができ、多相型に関して上記請求項１の発明と同等の効果を奏する。

【００９６】

請求項８記載の発明によれば、請求項６または請求項７の発明において、重複した不要な多相型のメンバ関数定義を取り除くことができると共に、不要な多相型メンバ関数の実体化を回避することができ、多相型に関して上記請求項１の発明と同等の効果を奏する。

【００９７】

請求項９記載の発明によれば、請求項６乃至請求項８の発明において、データ

または関数の定義の重複を防ぐことができ、全てのデータ型に関して上記請求項１の発明と同等の効果を奏する。

【００９８】

請求項１０記載の発明によれば、請求項６乃至請求項９の発明と同等の効果を奏する。

【図面の簡単な説明】

【図１】

本発明の第１実施形態に係るプログラム言語処理システムの基本構成を示すブロック図である。

【図２】

第１実施形態に係るプログラム言語処理システムの処理の流れを示す概念図である。

【図３】

第１実施形態に係る最適化処理を示すフローチャートである。

【図４】

第１実施形態に係る、多相型定義情報テーブルへの多相型定義の登録例を示す図である。

【図５】

第１実施形態に係る多相型定義の実体生成例を示す図である。

【図６】

第２実施形態に係る多相型関数定義の多相型定義情報テーブルへの登録例を説明するための図である。

【図７】

第２実施形態に係る多相型関数定義の実体生成例を示す図である。

【図８】

第３実施形態に係る多相型メンバ関数定義の多相型定義情報テーブル４ａへの登録例を示す図である。

【図９】

第３実施形態に係る多相型メンバ関数の最適化処理を示すフローチャートであ

る。

【図１０】

第３実施形態に係る多相型のメンバ関数定義の実体生成例を示す図である。

【図１１】

多相型の定義の従来例とその利用方法を示す図である。

【図１２】

実体化された多相型の定義の従来例を示す図である。

【図１３】

多相型の関数定義の従来例を示す図である。

【図１４】

従来のプログラム言語処理システムの構成を示すブロック図である。

【図１５】

従来のプログラム言語処理システムの処理の流れを示す概念図である。

【図１６】

従来の多相型の実現例を示す図である。

【図１７】

標準的なファイル構成における従来の多相型の定義方法例を示す図である。

【図１８】

多相型の関数定義をヘッダファイルに置いた従来例を示す図である。

【図１９】

多相型と同名のファイルに多相型の関数定義を置いた従来例を示す図である。

【図２０】

不必要な多相型のメンバ関数定義の実体が生成される従来例を示す図である。

【図２１】

複数のリロケータブルオブジェクトファイル中に多相型の関数定義の実体が重複して生成される例を示す図である。

【符号の説明】

１　ソフトウェア駆動装置

２　原始プログラム群

３　前処理装置

４　最適化処理装置

４ａ　多相型定義情報テーブル

５　言語処理装置

６　リンク処理装置

７　目的プログラム

８　前処理後のファイル

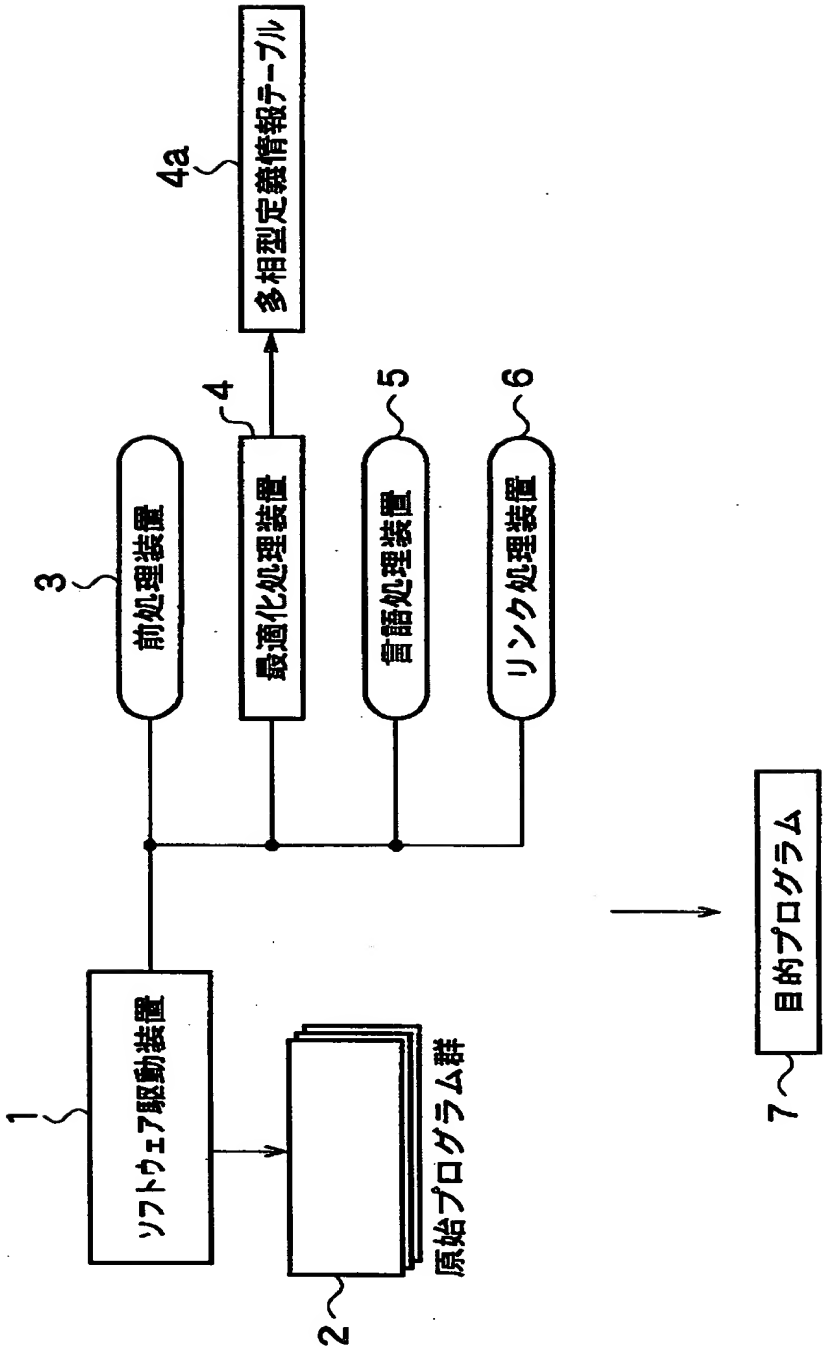
９　リロケータブルオブジェクト

ｆ０，ｆ１，ｆ２，ｆ３　前処理後の原始プログラム

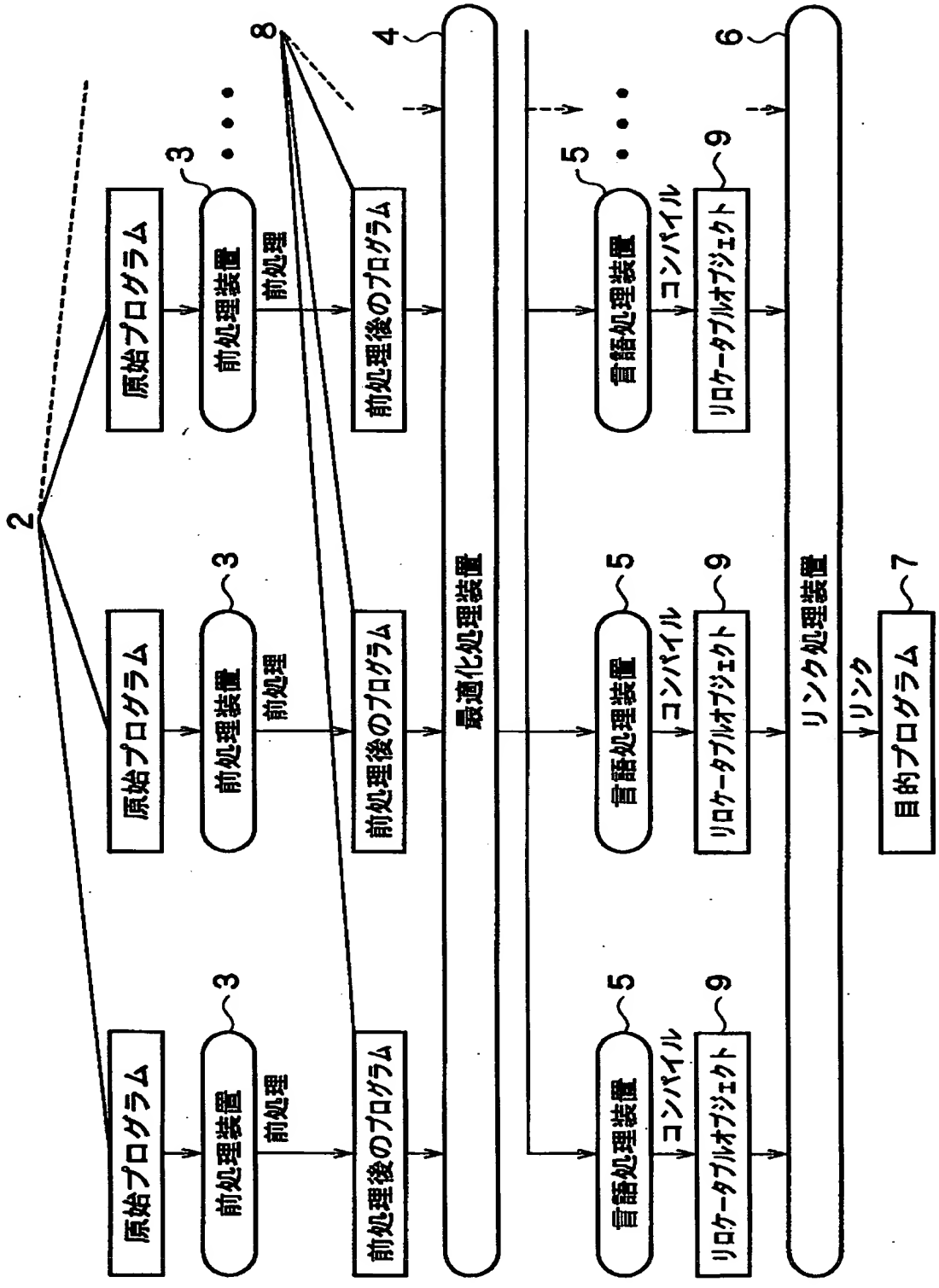
ｆ０’，ｆ１’，ｆ２’，ｆ３’　最適化後の原始プログラム

【書類名】　　図面

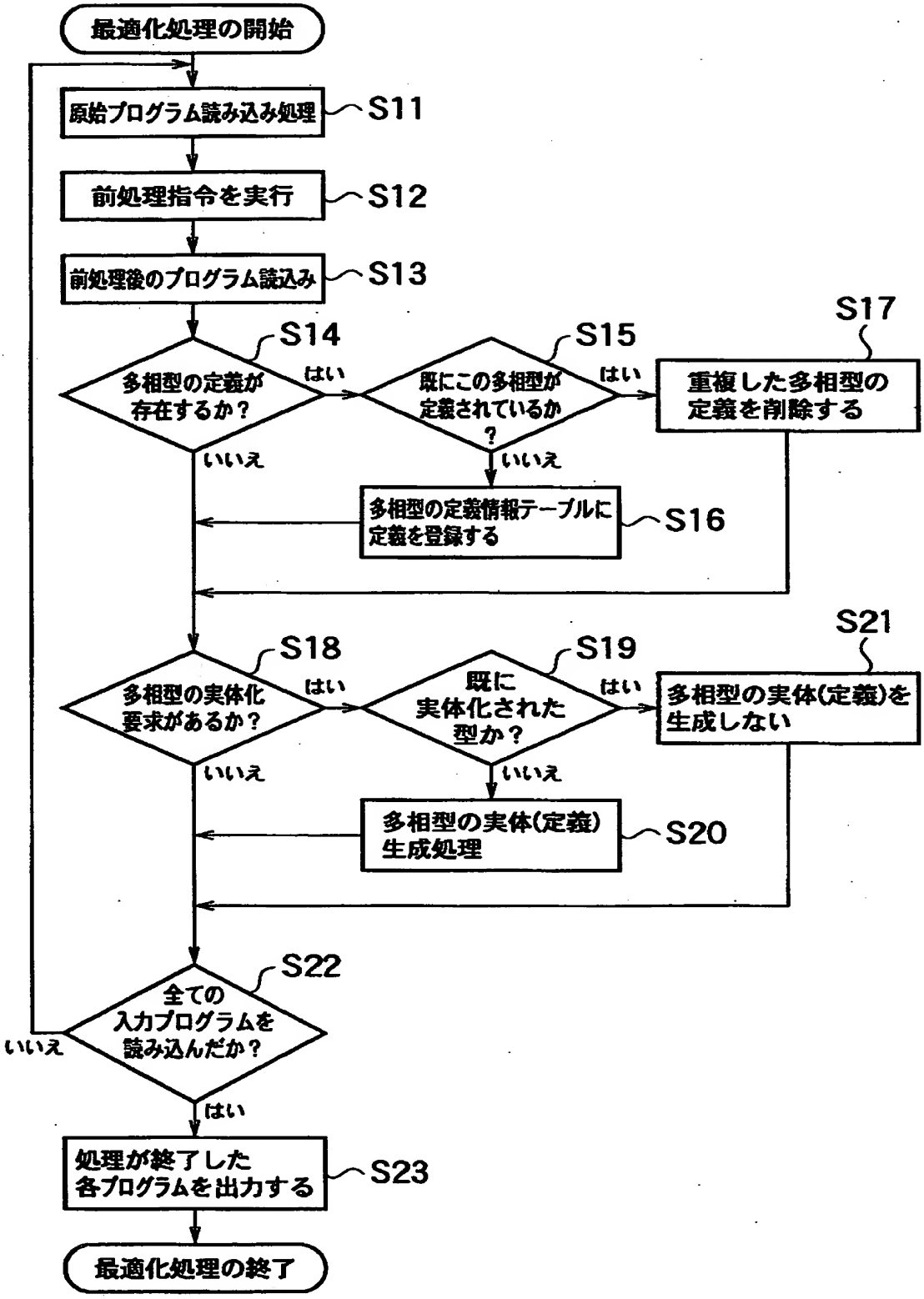
【図１】


ソフトウェア駆動装置
1
原始プログラム群
2
前処理装置
3
最適化処理装置
4
多相型定義情報テーブル
4a
言語処理装置
5
リンク処理装置
6
目的プログラム
7

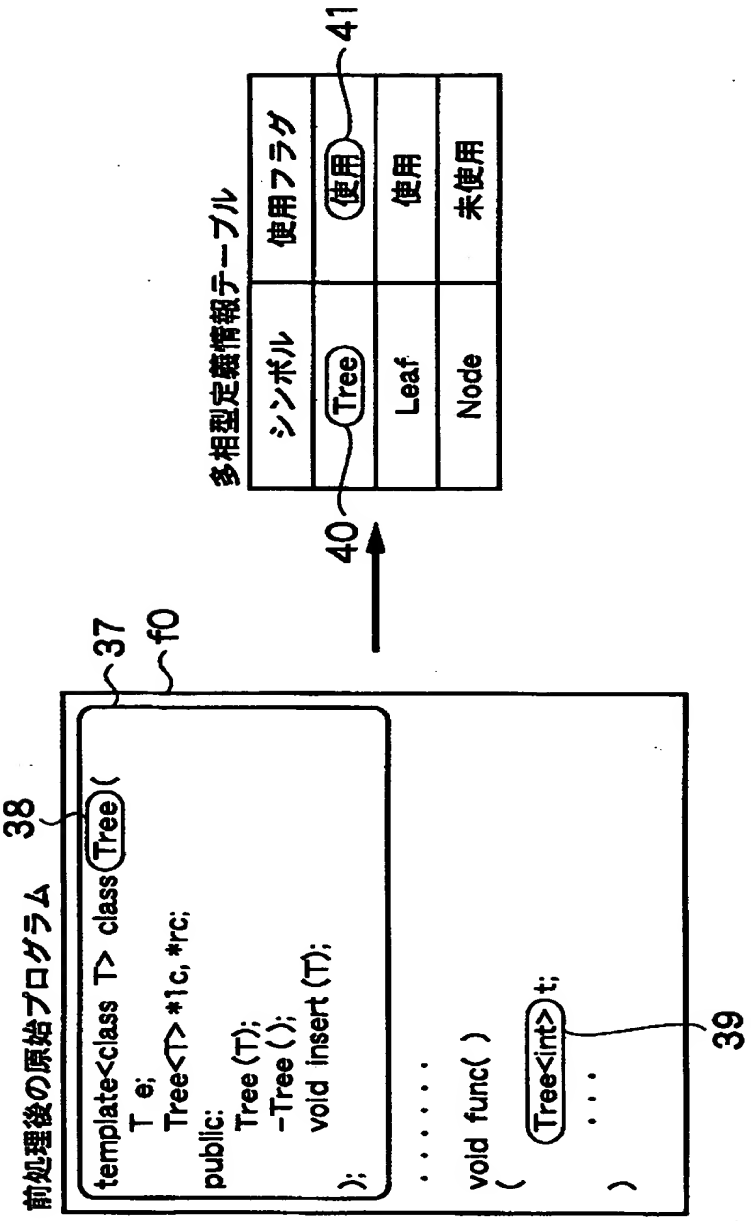
【図２】


2
原始プログラム
前処理装置
3
前処理
前処理後のプログラム
8
4
最適化処理装置
言語処理装置
5
コンパイル
リロケータブルオブジェクト
9
6
リンク処理装置
リンク
目的プログラム
7

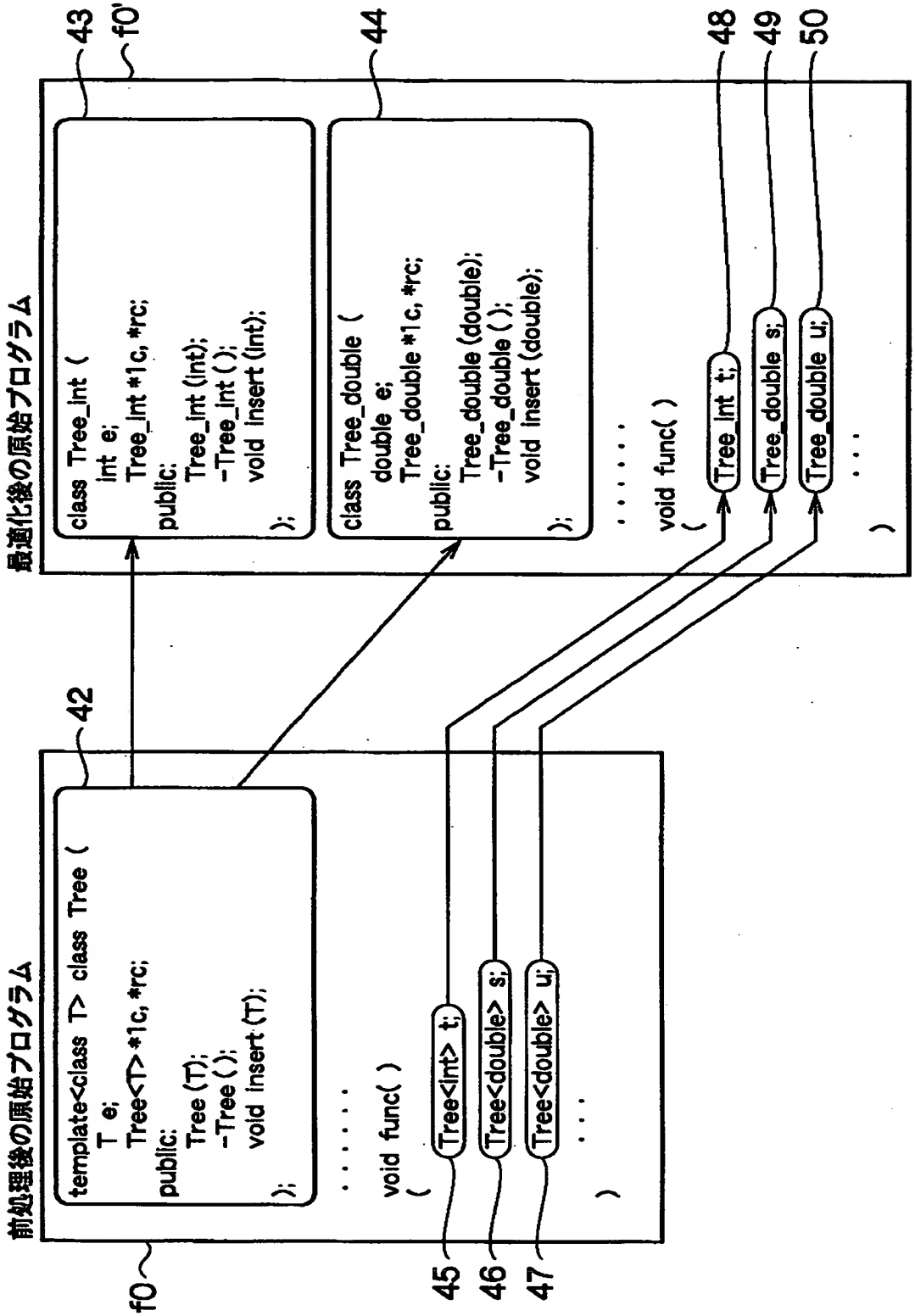
【図３】


最適化処理の開始
原始プログラム読み込み処理
S11
前処理指令を実行
S12
前処理後のプログラム読込み
S13
S14
多相型の定義が存在するか？
はい
S15
既にこの多相型が定義されているか？
はい
S17
重複した多相型の定義を削除する
いいえ
いいえ
多相型の定義情報テーブルに定義を登録する
S16
S18
多相型の実体化要求があるか？
はい
S19
既に実体化された型か？
はい
S21
多相型の実体(定義)を生成しない
いいえ
いいえ
多相型の実体(定義)生成処理
S20
S22
全ての入力プログラムを読み込んだか？
いいえ
はい
処理が終了した各プログラムを出力する
S23
最適化処理の終了

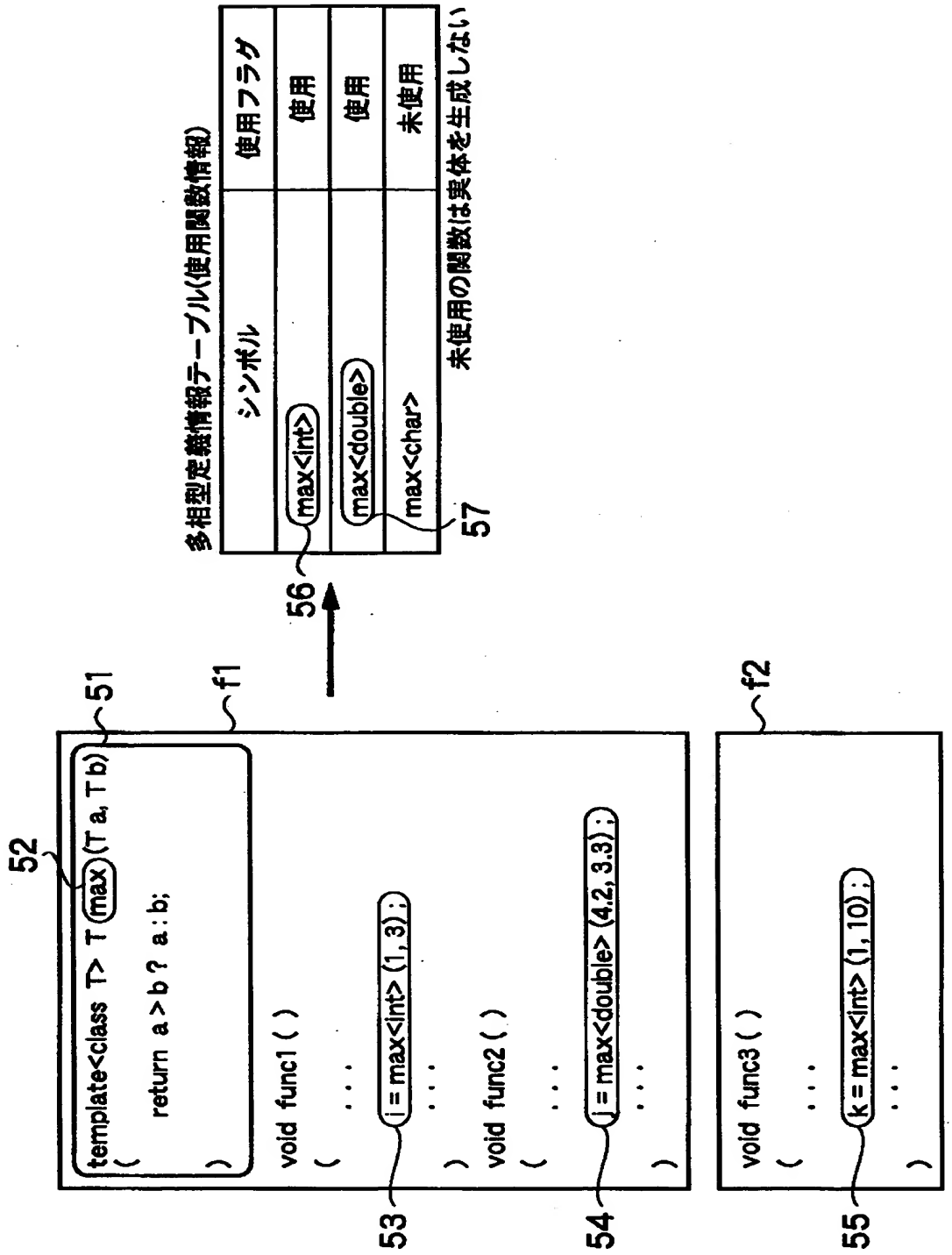
【図４】


前処理後の原始プログラム
38
template<class T> class Tree{
T e;
Tree<T> *lc, *rc;
public:
Tree (T);
~Tree ();
void insert (T);
};
37
f0
.....
void func( )
{
Tree<int> t;
...
}
39
40
多相型定義情報テーブル
41
シンボル
使用フラグ
Tree
使用
Leaf
使用
Node
未使用

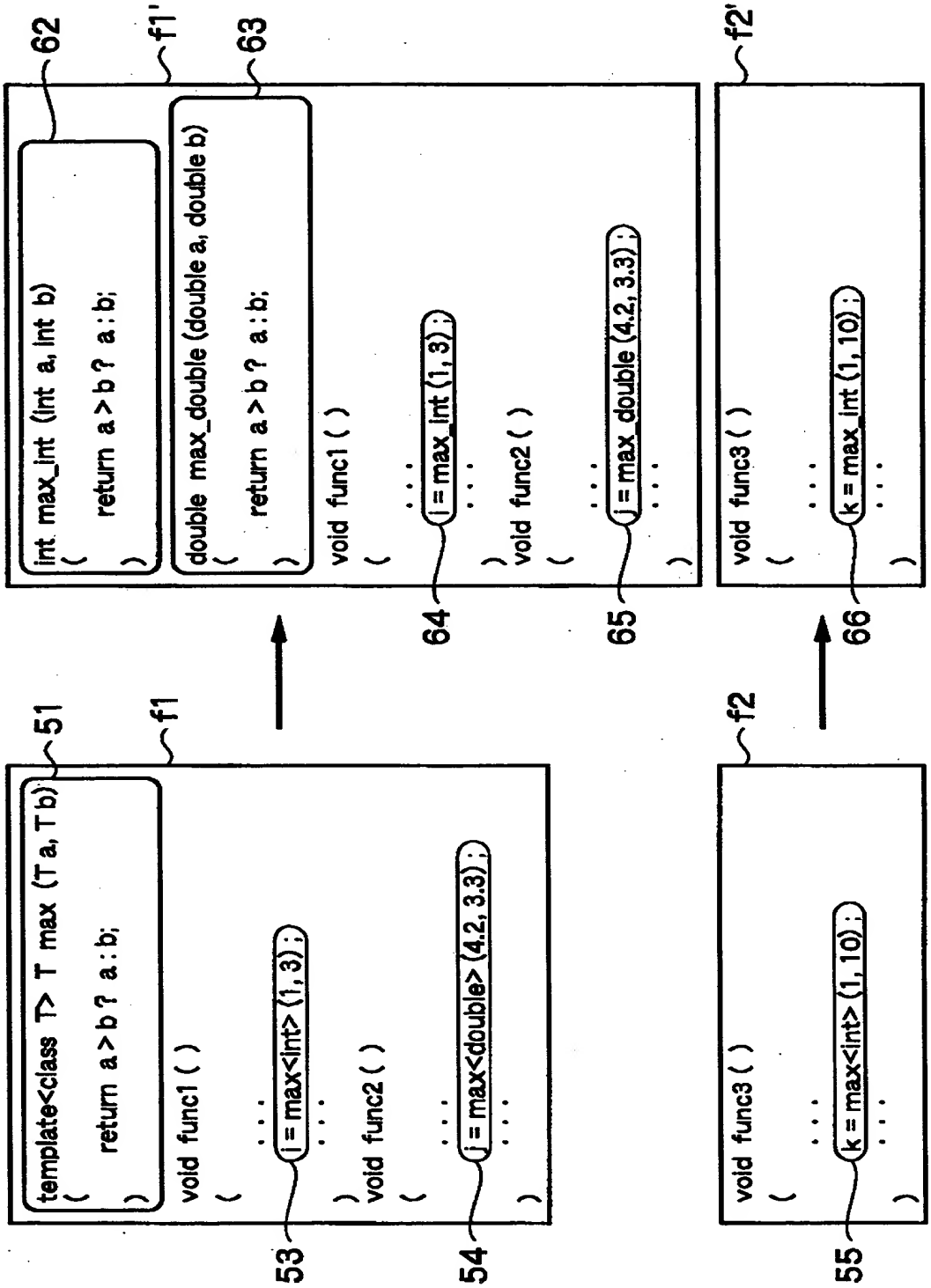
【図５】


前処理後の原始プログラム
template<class T> class Tree {
T e;
Tree<T> *lc, *rc;
public:
Tree (T);
~Tree ( );
void insert (T);
};
. . . . . .
void func( )
{
Tree<int> t;
Tree<double> s;
Tree<double> u;
. . .
}
f0
42
45
46
47
最適化後の原始プログラム
class Tree_int {
int e;
Tree_int *lc, *rc;
public:
Tree_int (int);
~Tree_int ( );
void insert (int);
};
class Tree_double {
double e;
Tree_double *lc, *rc;
public:
Tree_double (double);
~Tree_double ( );
void insert (double);
};
. . . . . .
void func( )
{
Tree_int t;
Tree_double s;
Tree_double u;
. . .
}
f0'
43
44
48
49
50

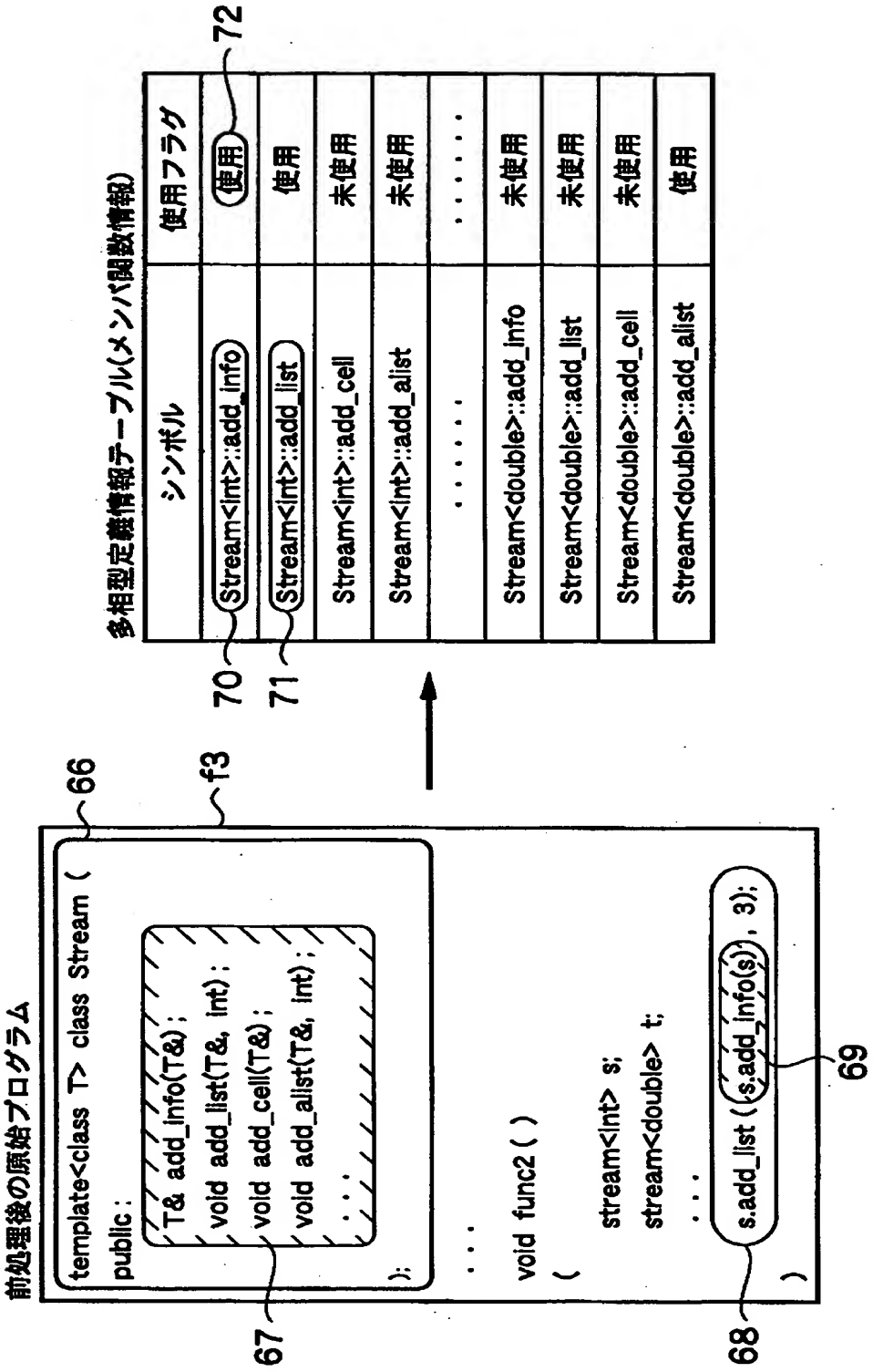
【図６】


多相型定義情報テーブル(使用関数情報)
シンボル
使用フラグ
max<int>
使用
max<double>
使用
max<char>
未使用
未使用の関数は実体を生成しない
56
57
51
52
f1
f2
template<class T> T max(T a, T b)
{
return a > b ? a : b;
}
void func1 ( )
{
...
i = max<int> (1, 3);
...
}
void func2 ( )
{
...
j = max<double> (4.2, 3.3);
...
}
void func3 ( )
{
...
k = max<int> (1, 10);
...
}
53
54
55

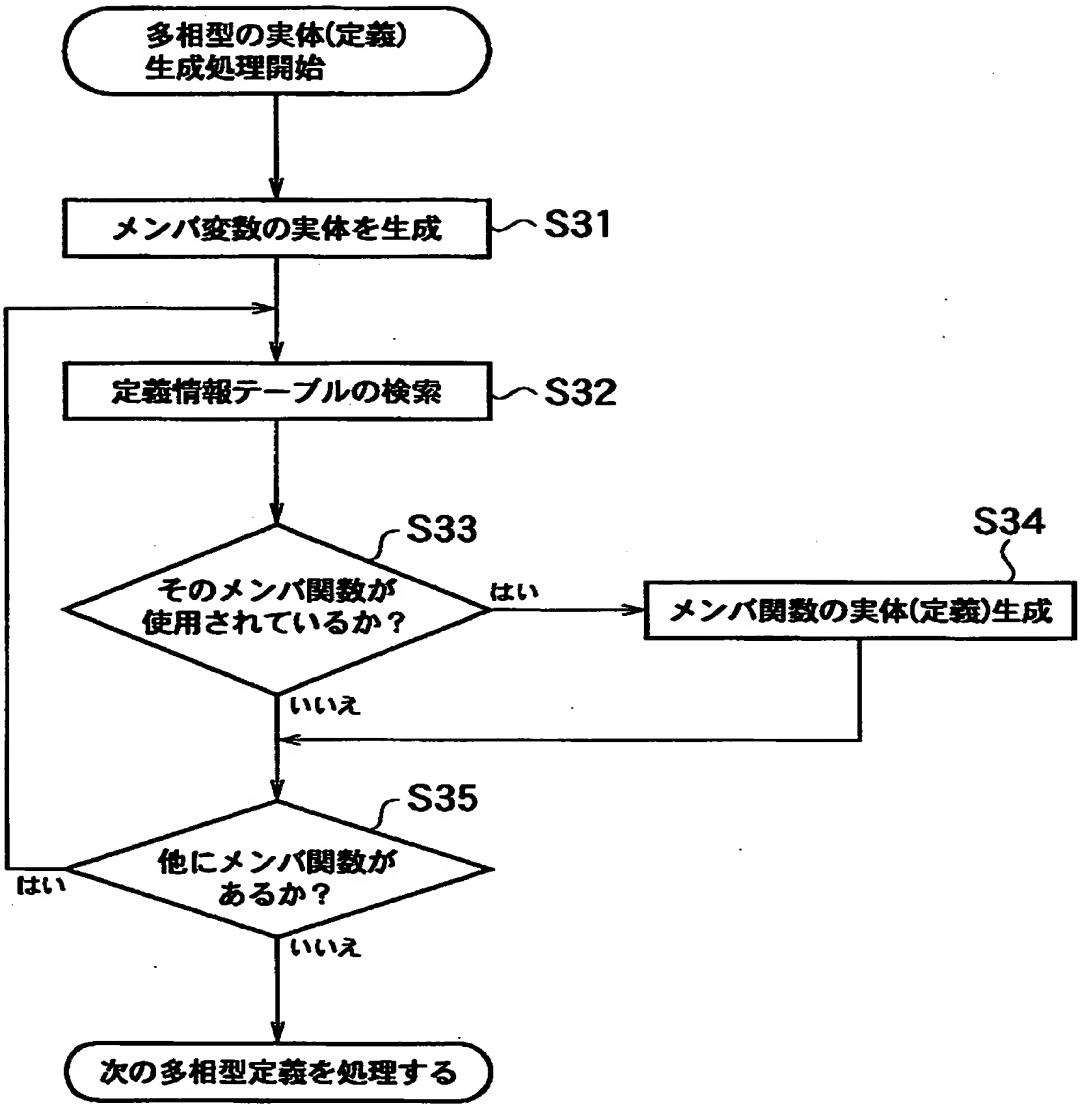
【図７】


62
f1'
63
f2'
int max_int (int a, int b)
{
return a > b ? a : b;
}
double max_double (double a, double b)
{
return a > b ? a : b;
}
void func1 ( )
{
...
i = max_int (1, 3) ;
...
}
void func2 ( )
{
...
j = max_double (4.2, 3.3) ;
...
}
void func3 ( )
{
...
k = max_int (1, 10) ;
...
}
64
65
66
51
f1
f2
template<class T> T max (T a, T b)
{
return a > b ? a : b;
}
void func1 ( )
{
...
i = max<int> (1, 3) ;
...
}
void func2 ( )
{
...
j = max<double> (4.2, 3.3) ;
...
}
void func3 ( )
{
...
k = max<int> (1, 10) ;
...
}
53
54
55

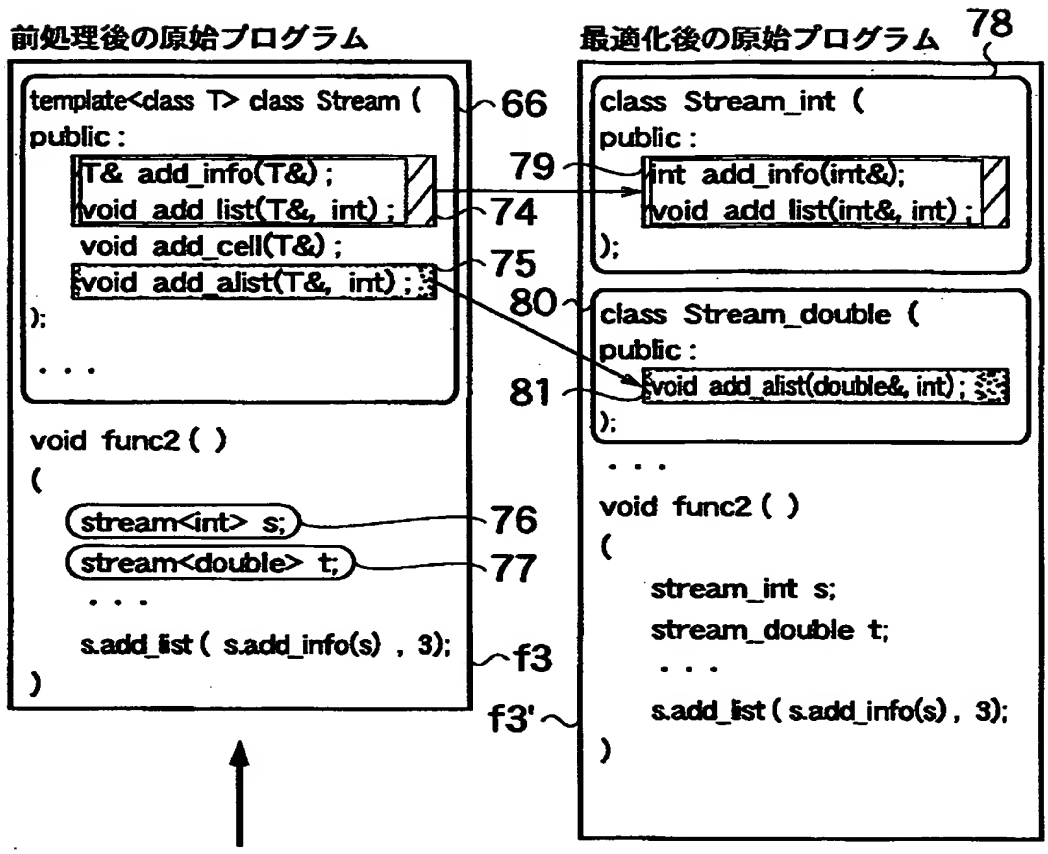
【図８】


前処理後の原始プログラム
template<class T> class Stream {
public:
T& add_info(T&);
void add_list(T&, int);
void add_cell(T&);
void add_alist(T&, int);
...
};
...
void func2 ( )
{
stream<int> s;
stream<double> t;
...
s.add_list ((s.add_info(s)), 3);
}
66
f3
67
68
69
多相型定義情報テーブル(メンバ関数情報)
シンボル
使用フラグ
Stream<int>::add_info
使用
Stream<int>::add_list
使用
Stream<int>::add_cell
未使用
Stream<int>::add_alist
未使用
Stream<double>::add_info
未使用
Stream<double>::add_list
未使用
Stream<double>::add_cell
未使用
Stream<double>::add_alist
使用
70
71
72

【図９】


多相型の実体(定義)
生成処理開始
メンバ変数の実体を生成
S31
定義情報テーブルの検索
S32
S33
そのメンバ関数が
使用されているか？
はい
S34
メンバ関数の実体(定義)生成
いいえ
S35
他にメンバ関数が
あるか？
はい
いいえ
次の多相型定義を処理する

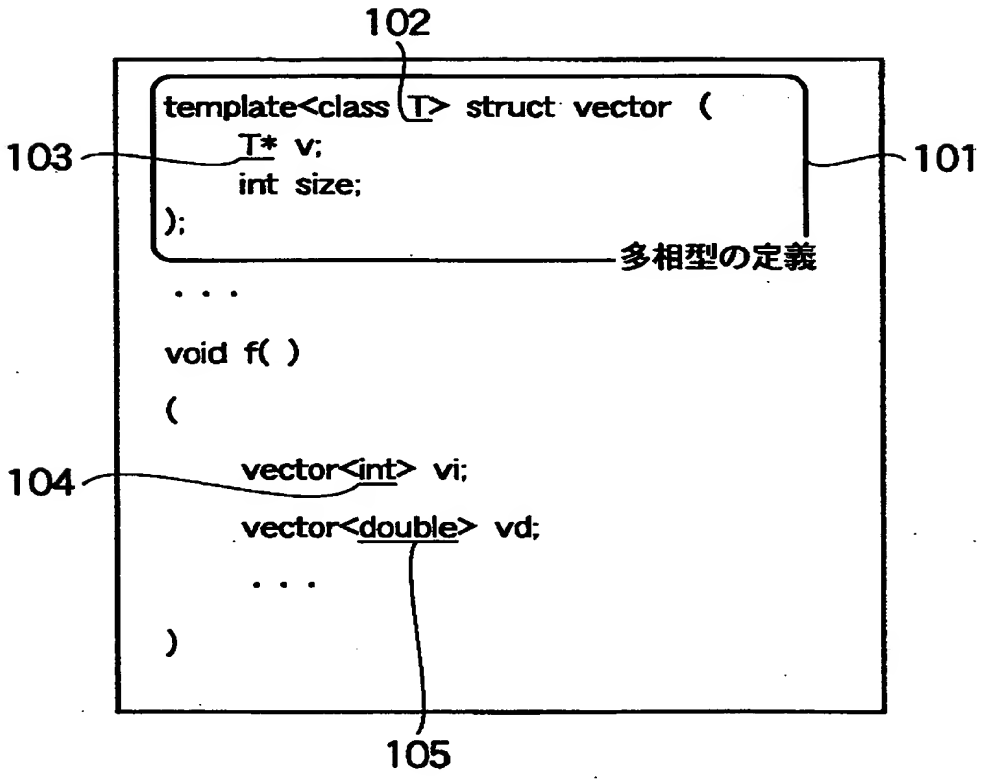
【図１０】



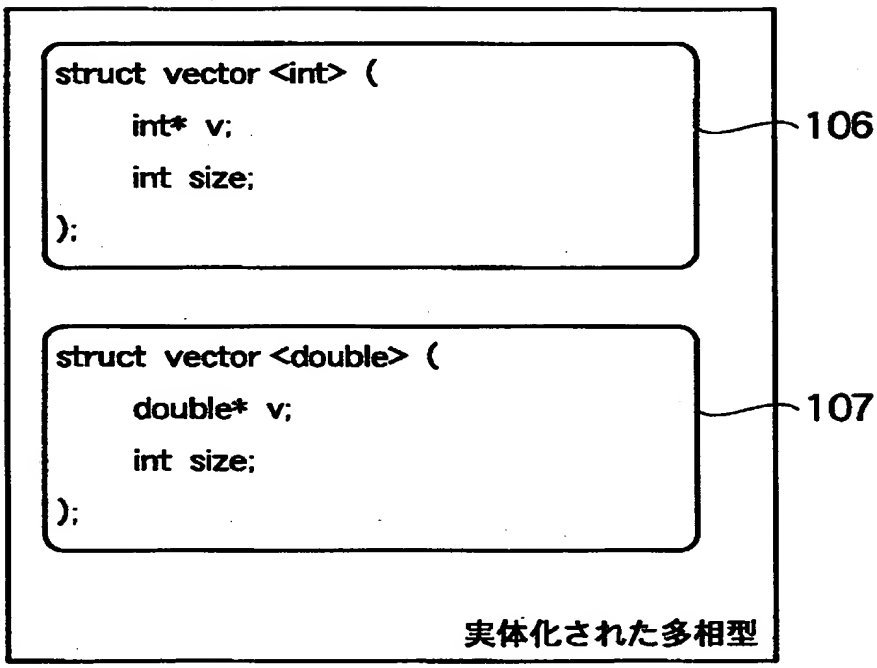
多相型定義情報テーブル(メンバ関数情報)

シンボル	使用フラグ
Stream<int>::add_info	使用
Stream<int>::add_list	使用
Stream<int>::add_cell	未使用
Stream<int>::add_alist	未使用
Stream<double>::add_info	未使用
Stream<double>::add_list	未使用
Stream<double>::add_cell	未使用
Stream<double>::add_alist	使用

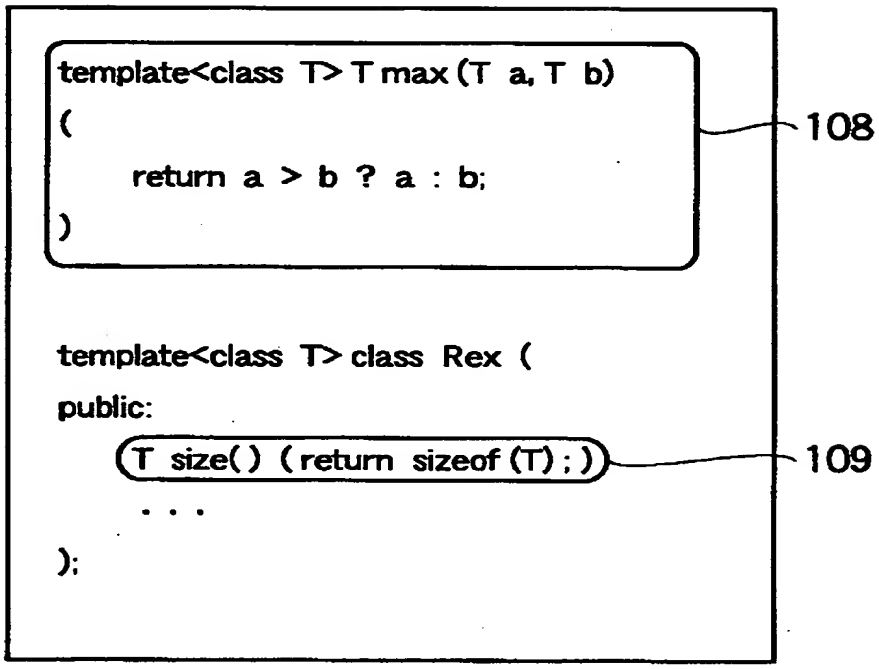
【図１１】


102
template<class T> struct vector (
T* v;
int size;
);
多相型の定義
103
101
. . .
void f( )
(
104
vector<int> vi;
vector<double> vd;
. . .
)
105

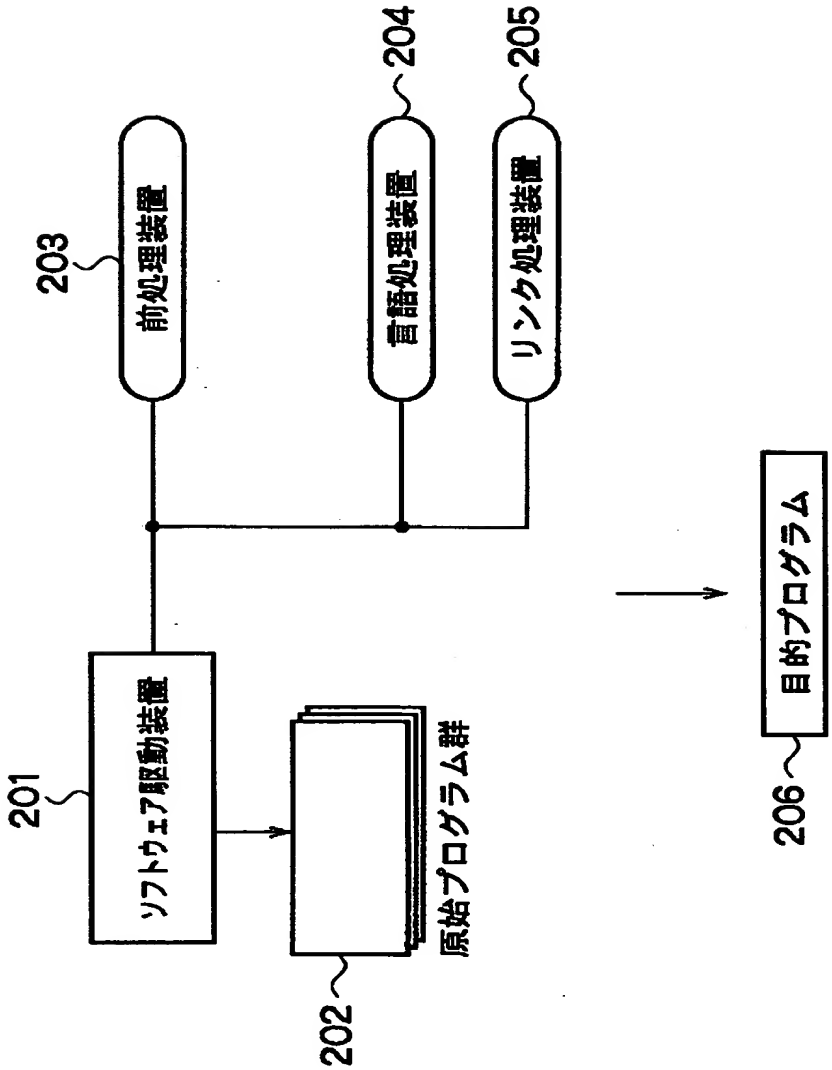
【図１２】


struct vector <int> (
int* v;
int size;
);
106
struct vector <double> (
double* v;
int size;
);
107
実体化された多相型

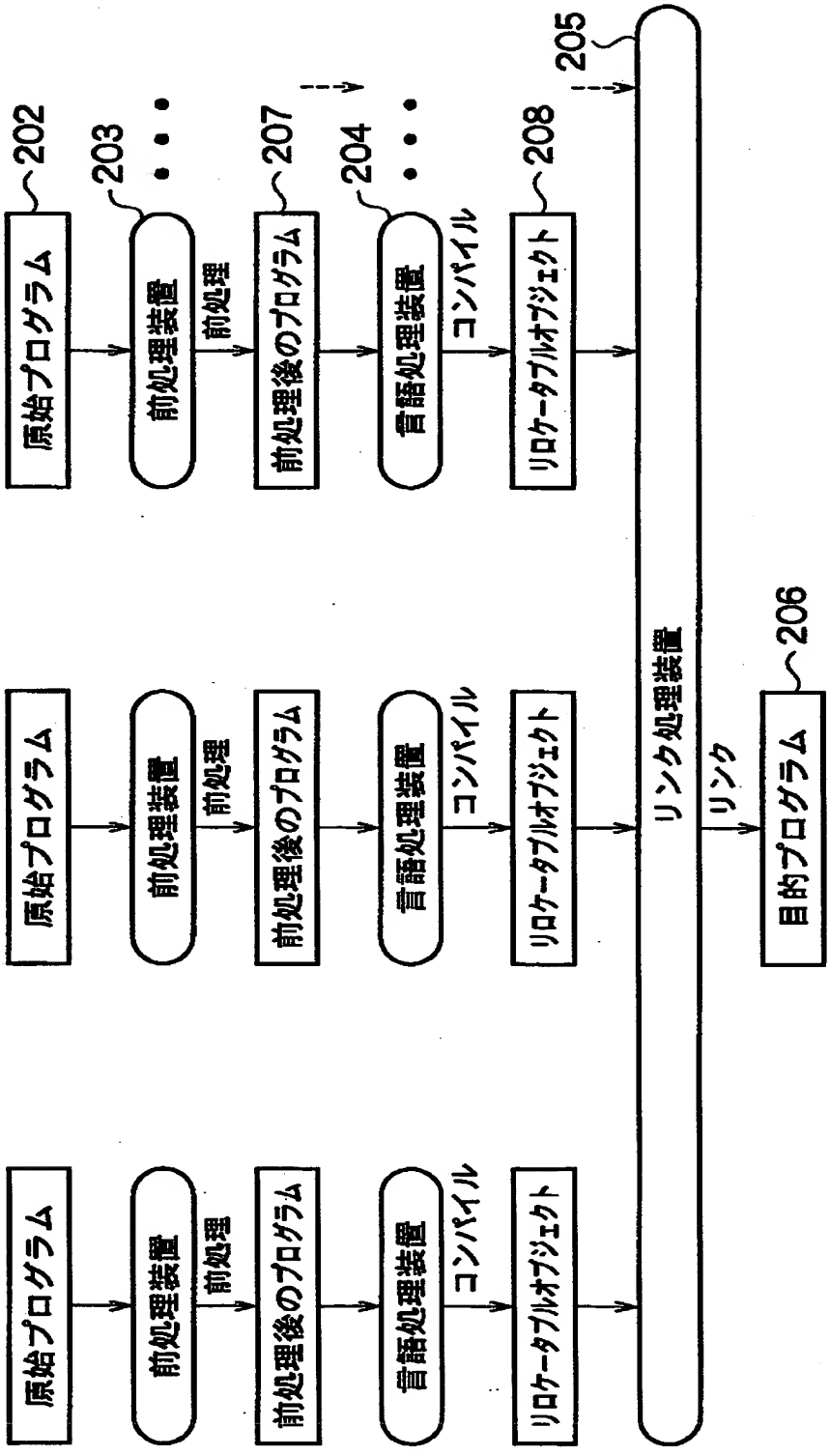
【図１３】


template<class T> T max (T a, T b)
(
return a > b ? a : b;
)
108
template<class T> class Rex (
public:
T size( ) (return sizeof (T) ;)
109
. . .
);

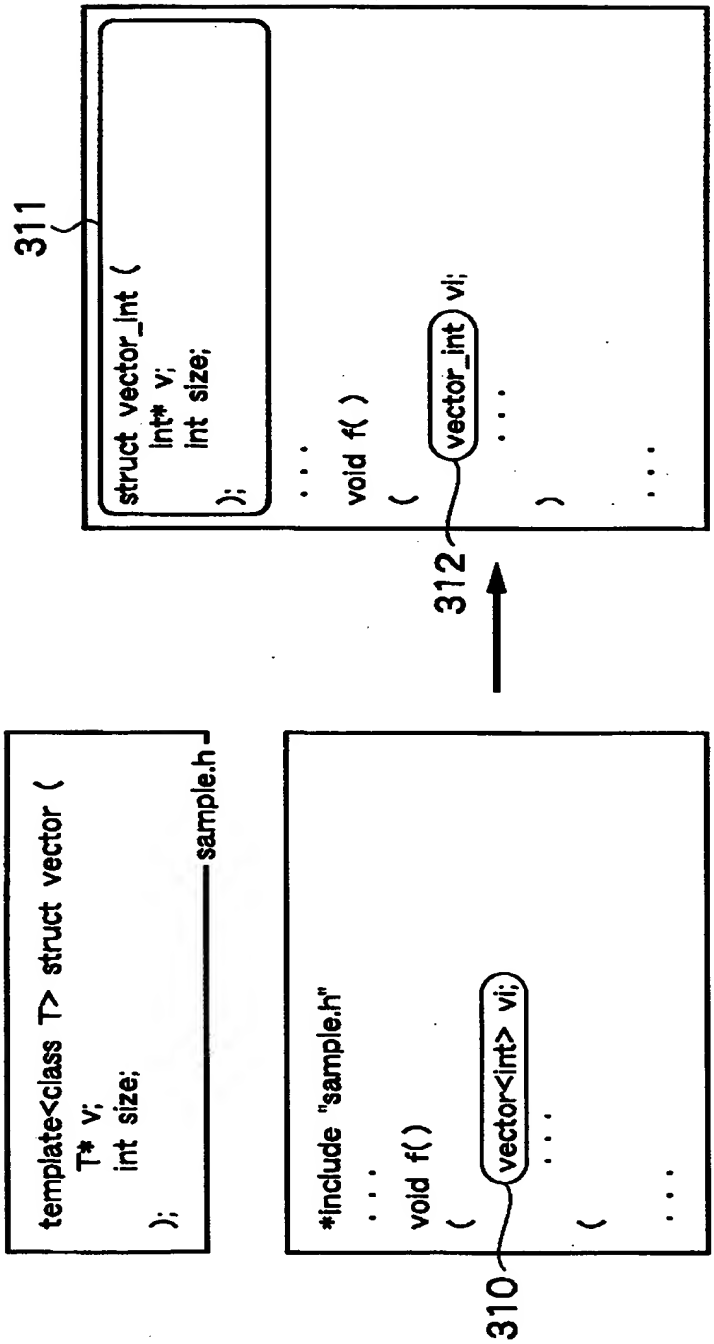
【図１４】


203
前処理装置
204
言語処理装置
205
リンク処理装置
201
ソフトウェア駆動装置
202
原始プログラム群
206
目的プログラム

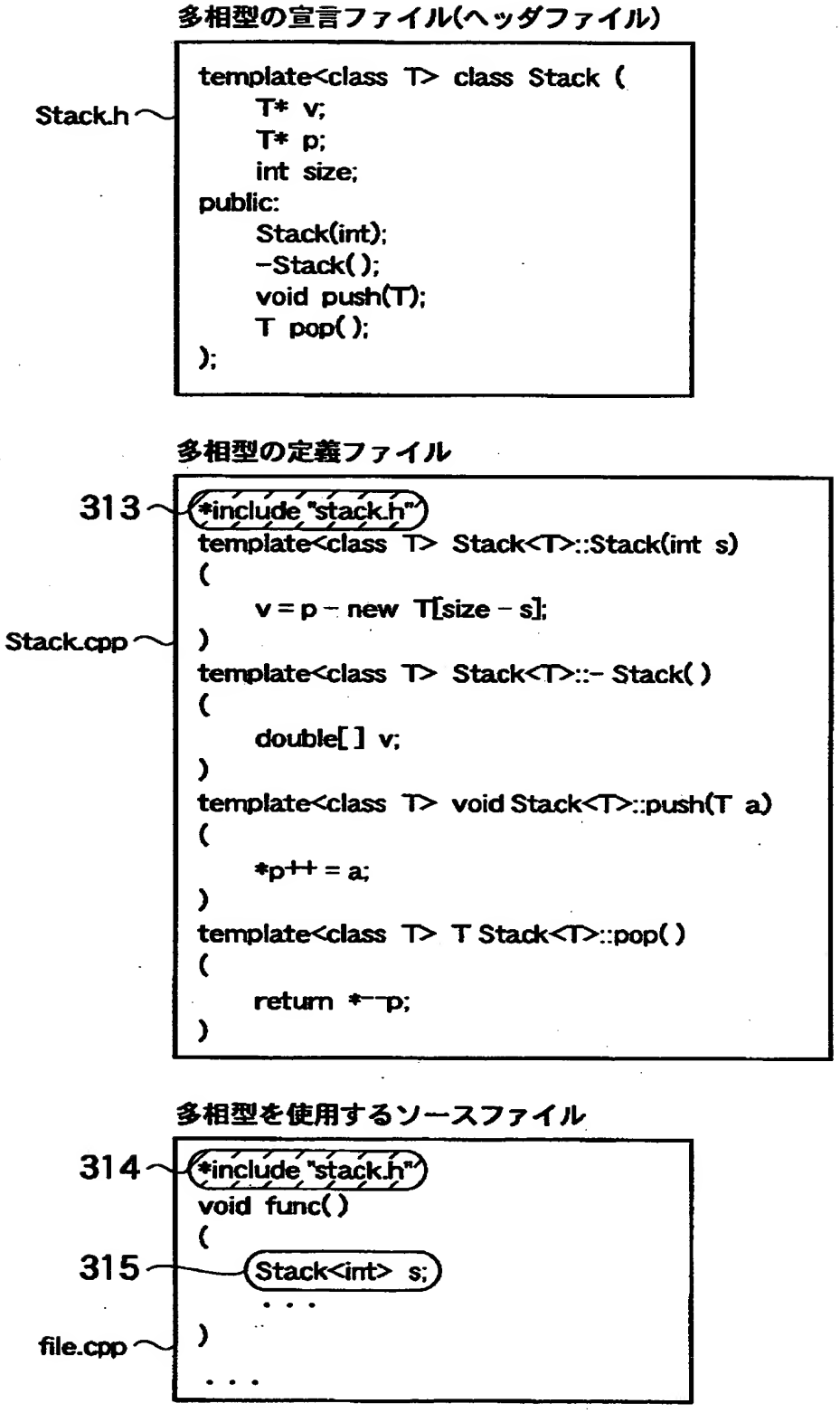
【図１５】


原始プログラム
202
前処理装置
203
前処理
前処理後のプログラム
207
言語処理装置
204
コンパイル
リロケータブルオブジェクト
208
205
リンク処理装置
リンク
目的プログラム
206

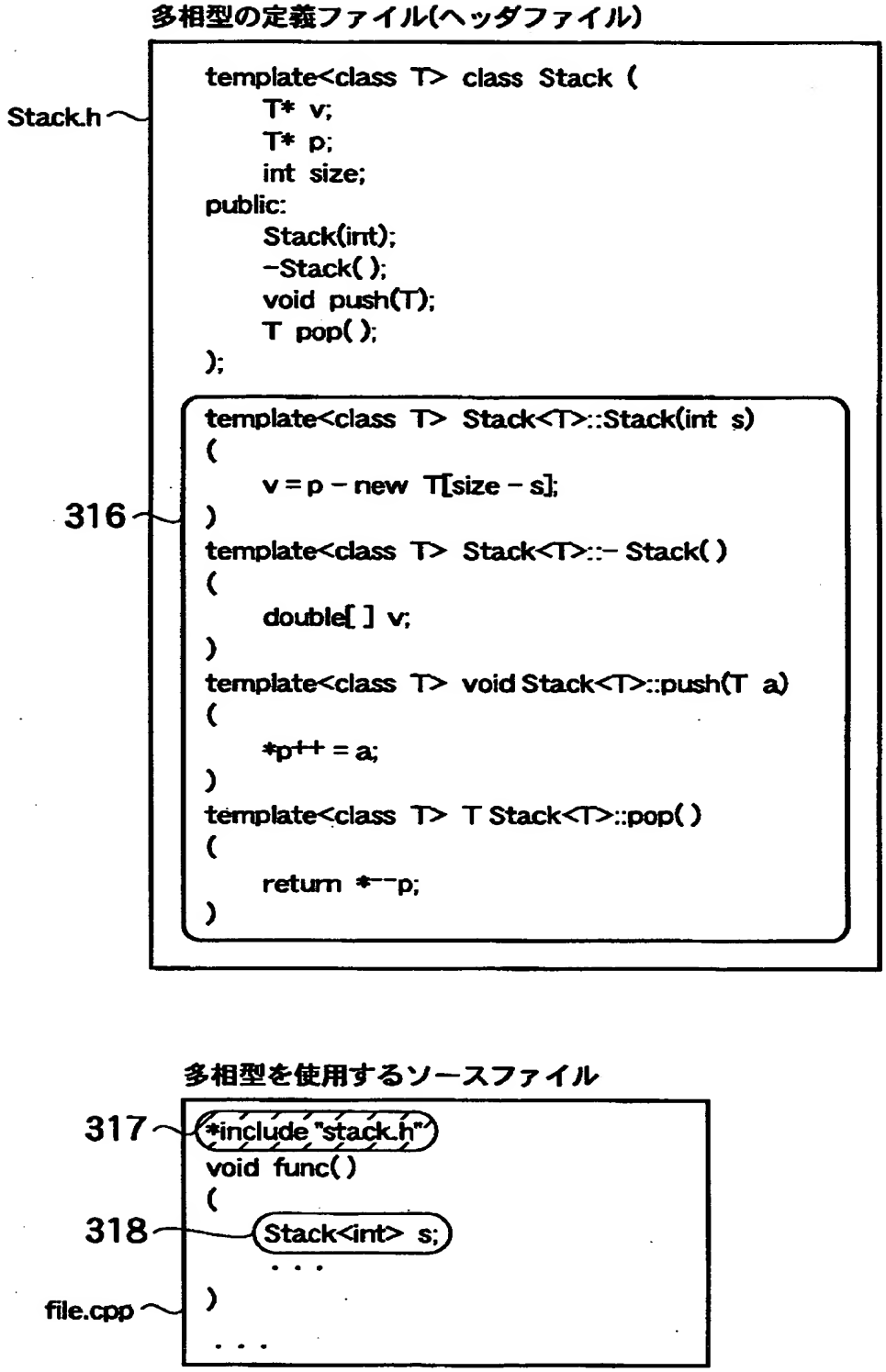
【図１６】


311
struct vector_int (
int* v;
int size;
);
...
void f( )
(
vector_int vi;
...
)
...
312
template<class T> struct vector (
T* v;
int size;
);
sample.h
#include "sample.h"
...
void f()
(
vector<int> vi;
...
(
...
310

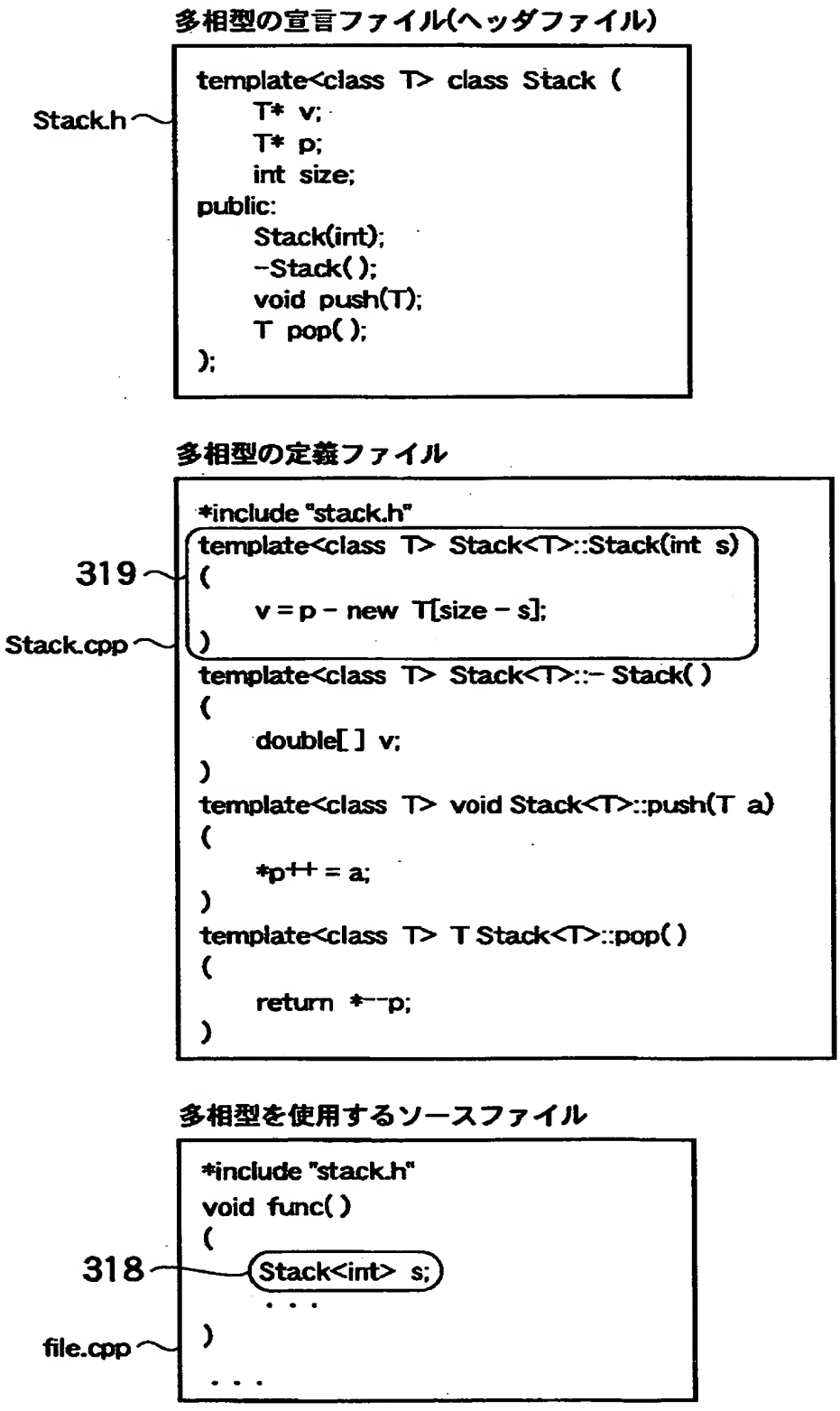
【図１７】


多相型の宣言ファイル(ヘッダファイル)
Stack.h
template<class T> class Stack (
T* v;
T* p;
int size;
public:
Stack(int);
~Stack( );
void push(T);
T pop( );
);
多相型の定義ファイル
313
#include "stack.h"
template<class T> Stack<T>::Stack(int s)
(
v = p = new T[size - s];
)
Stack.cpp
template<class T> Stack<T>::~ Stack( )
(
double[ ] v;
)
template<class T> void Stack<T>::push(T a)
(
*p++ = a;
)
template<class T> T Stack<T>::pop( )
(
return *--p;
)
多相型を使用するソースファイル
314
#include "stack.h"
void func( )
(
315
Stack<int> s;
. . .
)
file.cpp
. . .

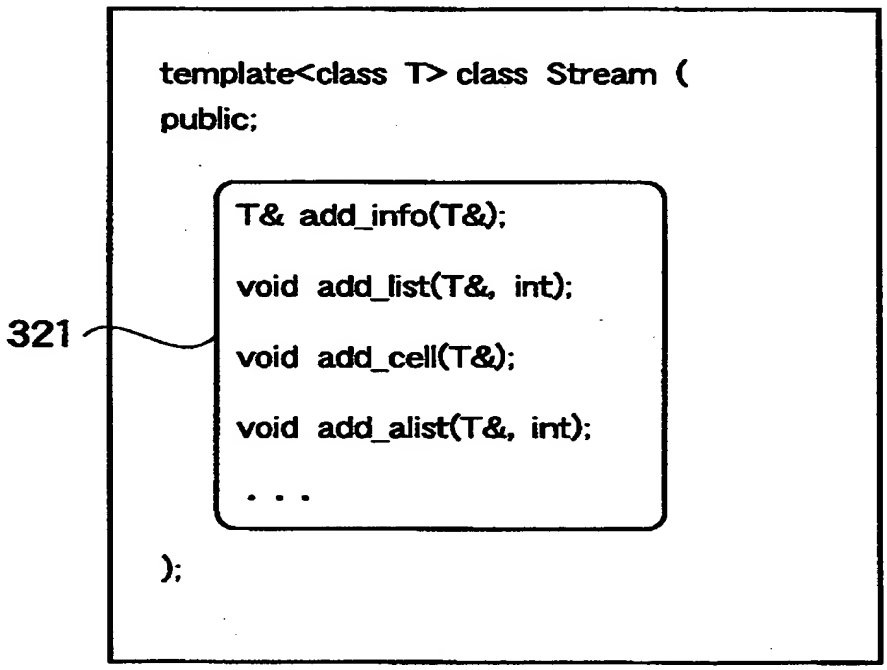
【図１８】


多相型の定義ファイル(ヘッダファイル)
Stack.h
template<class T> class Stack (
T* v;
T* p;
int size;
public:
Stack(int);
~Stack( );
void push(T);
T pop( );
);
template<class T> Stack<T>::Stack(int s)
(
v = p = new T[size - s];
316
)
template<class T> Stack<T>::~ Stack( )
(
double[ ] v;
)
template<class T> void Stack<T>::push(T a)
(
*p++ = a;
)
template<class T> T Stack<T>::pop( )
(
return *--p;
)
多相型を使用するソースファイル
317
#include "stack.h"
void func( )
(
318
Stack<int> s;
. . .
)
file.cpp
. . .

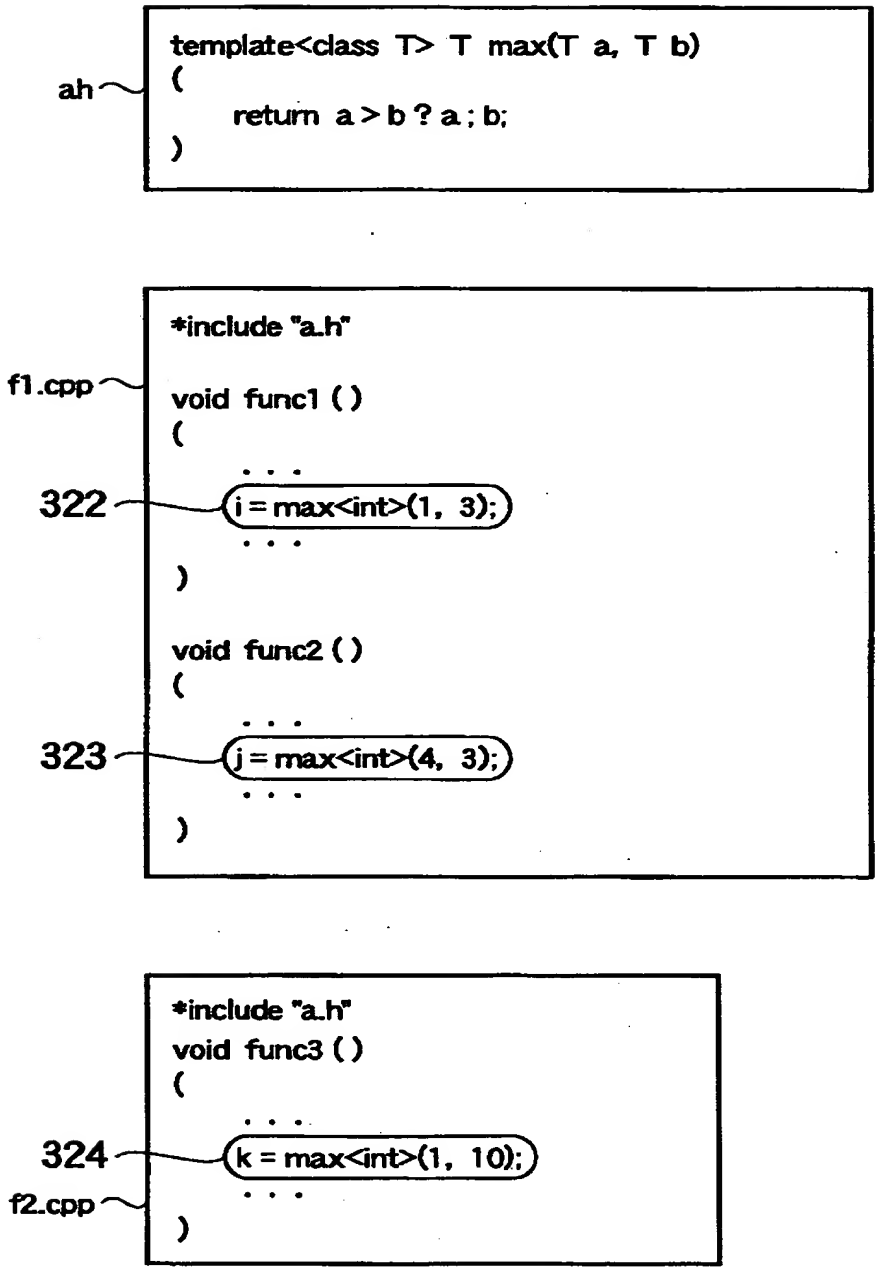
【図１９】


多相型の宣言ファイル(ヘッダファイル)
Stack.h
template<class T> class Stack (
T* v;
T* p;
int size;
public:
Stack(int);
~Stack( );
void push(T);
T pop( );
);
多相型の定義ファイル
319
Stack.cpp
*include "stack.h"
template<class T> Stack<T>::Stack(int s)
(
v = p - new T[size - s];
)
template<class T> Stack<T>::~ Stack( )
(
double[ ] v;
)
template<class T> void Stack<T>::push(T a)
(
*p++ = a;
)
template<class T> T Stack<T>::pop( )
(
return *--p;
)
多相型を使用するソースファイル
318
file.cpp
*include "stack.h"
void func( )
(
Stack<int> s;
. . .
)
. . .

【図２０】


template<class T> class Stream (
public;
T& add_info(T&);
void add_list(T&, int);
void add_cell(T&);
void add_alist(T&, int);
. . .
);
321

【図２１】


ah
template<class T> T max(T a, T b)
(
return a > b ? a : b;
)
f1.cpp
#include "a.h"
void func1 ( )
(
322
i = max<int>(1, 3);
)
void func2 ( )
(
323
j = max<int>(4, 3);
)
#include "a.h"
void func3 ( )
(
324
k = max<int>(1, 10);
f2.cpp
)

【書類名】　　要約書

【要約】

【課題】　重複した不要な多相型の定義を取り除くことができると共に、不要な多相型の実体化を回避することができるプログラム言語処理システムのコード最適化方法を提供する。

【解決手段】　最適化処理装置４は、目的プログラム７を生成するために構成される全ての原始プログラム２に対し、多相型の情報を多相型定義情報テーブル４ａに格納し、この多相型定義情報テーブル４ａを参照して不要なコードを検知し、必要な定義のみ実体化し、各々の翻訳単位として言語処理装置５にプログラムを出力する。

【選択図】　　図１

出　願　人　履　歴　情　報

識別番号　　〔０００００３０７８〕

１．変更年月日　　１９９０年　８月２２日
［変更理由］　新規登録
住　所　神奈川県川崎市幸区堀川町７２番地
氏　名　株式会社東芝